SÓLO

PROGRAMADORES

Revista especializada para usuarios de PC

AÑO 2 Nº 9
1250 PTAS.



DE REGALO
El libro "LINUX, Unix para todos"

Resultado del 1er Torneo de Programación "Las 4 en Raya"

Tratamiento Digital de imagen

Y además:

- Fichas de programación
- Curso de ensamblador
- Sistemas expertos
- Reconocimiento de voz
- Clarion para Windows
- Curso de C++

Especial sobre compresión fractal de imágenes



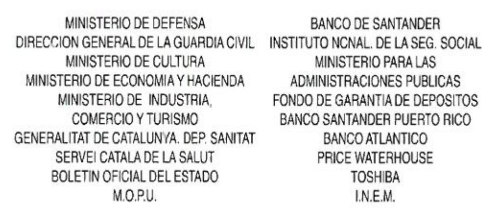












MINISTERIO DE DEFENSA
DIRECCION GENERAL DE LA GUARDIA CIVIL
MINISTERIO DE CULTURA
MINISTERIO DE ECONOMIA Y HACIENDA
MINISTERIO DE INDUSTRIA, COMERCIO Y TURISMO
GENERALITAT DE CATALUNYA. DEP. SANITAT
SERVEI CATALA DE LA SALUT
BOLETIN OFICIAL DEL ESTADO
M.O.P.U.

BANCO DE SANTANDER
INSTITUTO NCNAL. DE LA SEG. SOCIAL
MINISTERIO PARA LAS ADMINISTRACIONES PUBLICAS
FONDO DE GARANTIA DE DEPOSITOS
BANCO SANTANDER PUERTO RICO
BANCO ATLANTICO
PRICE WATERHOUSE
TOSHIBA
I.N.E.M.

SANTANDER NATIONAL BANK
CORPORACION FINANCIERA HISPAMER
ANALISTAS FINANCIEROS INTERNACIONALES
SEGUROS OCASO
EUROSEGUROS BBV
SEGUROS ATHENA
FYCSA (ALCATEL)
SECRETARIA DE ESTADO PARA LAS COMUNIDADES EUROPEAS

CONSTRUCCIONES AERONAUTICAS (CASA)
PATENTES TALGO
SINTEL
AVIACO
TRANSMEDITERRANEA
PUERTOS DEL ESTADO
GEC ALSTHOM
L´OREAL
NATIONALE NEDERLANDEN

FASA RENAULT
RED DE CONCESIONARIOS RENAULT
REPSOL EXPLORACION
REFINERIA DE GIBRALTAR
PETROGAL
ONDA CERO RADIO
TELEMADRID
ANTENA 3 TELEVISION
A.B.B. (ASEA BROWN BOVERI)
IVECO - PEGASO









TELECINCO (GESTEVISION-TELECINCO)
UNIVERSIDAD AUTONOMA DE MADRID
UNIVERSIDAD CARLOS III DE MADRID
INSTITUTO DE EMPRESA
TELEFONICA SISTEMAS*
S.P. EDITORES*
OLIVETTI*
SOFTWARE AG
JUNTA DE EXTEMADURA

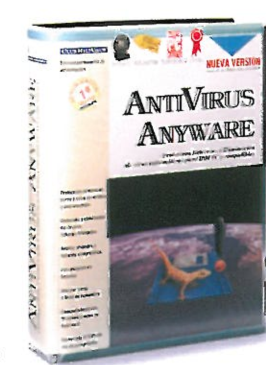
IBM*
SIEMENS NIXDORF*
DIGITAL*
BULL ESPAÑA*
FUJITSU*
INVESTRONICA*
SOFTWARE DE DIAGNOSTICO*
SEUR
CAJA LABORAL POPULAR

TELEFONICA
INFORMATICA EL CORTE INGLES*
ACTION*
GTI*
ALCATEL SISTEMAS DE INFORMACION*
INDAS
DELOITTE & TOUCHE

EL CORTE INGLES*
SOFTWARE DE ESPAÑA*
INFORMIX
ABBOTT CIENTIFICA
ALBILUX
ELIDA GIBBS
OXFORD UNIVERSITY PRESS
A.E.N.A. (AEROPUERTOS ESPAÑOLES Y NAVEGACION AEREA)

9 de cada 10 Ordenadores prefieren ANTIVIRUS ANYWARE.

El resultado está a la vista. O, ¿cree que tantas Empresas pueden estar equivocadas?. Tienen razones para no estarlo.



1. MAYOR PROTECCION.
Incorpora un Sistema de Protección con una ocupación mínima en RAM. Detecta el Virus antes de que pueda introducirse e impide el uso del fichero contaminado, avisándole a través de una ventana de alarma.

2. DETECCION INSTANTANEA.
De forma rápida y precisa, analiza cualquier unidad, directorio o fichero. Comprueba si su disco duro o disquetes contienen algún Virus. Chequea ficheros (incluso comprimidos), sector de arranque, tabla de particiones y unidades de red.

3. ELIMINACION DEFINITIVA.
Elimina los Virus allí donde se encuentren, sin dañar los ficheros.

4. ACTUALIZACION PERMANENTE.
Usted puede recibir las nuevas versiones cómodamente en su domicilio o capturarlas vía modem.

ANYWARE pone a su disposición el CLUB DE USUARIOS HELP VIRUS con una HOT LINE exclusiva para consultas, noticias, BBS, etc.


PREMIO PC MAGAZINE MEJOR UTILIDAD 1991



RECOMENDACION PCWORLD MEJOR PRODUCTO ANTIVIRUS

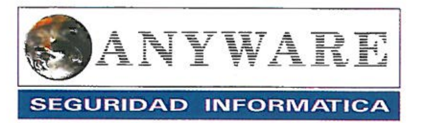


RECOMENDACION PCMAGAZINE MEJOR PRODUCTO ANTIVIRUS


CATACOM 93 MEJOR ANTIVIRUS



PREMIO PC MAGAZINE MEJOR UTILIDAD 1993



Orense, 36 3º. 28020 MADRID. España.
Tel.: (91) 556 92 15. Fax.: (91) 556 14 04.

EMPRESAS E INSTITUCIONES CON LICENCIA CORPORATIVA DE ANTIVIRUS ANYWARE
(*)EMPRESAS DISTRIBUIDORAS DE ANTIVIRUS ANYWARE.



CARTA DEL DIRECTOR



ALTO O BAJO NIVEL

Durante años ha existido la división entre los programadores de alto y bajo nivel. Los primeros, tildados por los segundos de *antiestructurados,* con esa aureola que rodeaba a los pioneros *geniecillos* informáticos, han sobrevivido en muchas empresas impertérritos al paso del tiempo. En nuestros días, los programadores de alto nivel son la inmensa mayoría mientras que los expertos en ensamblador han quedado relegados a un segundo plano. Los motivos, de sobra conocidos por todos, no son el objeto de este artículo editorial. Sin embargo y a pesar de muchas flechas apunten en una dirección, el hecho es que el experto en *Turbo Debugger*, capaz de encontrar cualquier error, ocupa en todas las empresas un lugar privilegiado y todos intuyen que es la última persona de la que la empresa prescindirá. Curiosa paradoja ¿no? Tal vez sea ello el motivo por el cual, a pesar de que los lenguajes de programación sean cada vez más potentes y necesiten menor conocimiento del hardware, la gente siga aprendiendo lenguajes de bajo nivel. Por eso en SP publicamos un curso de código máquina así como artículos esporádicos en los que se tratan temas concretos desde bajo nivel en profundidad, para que la gente siga aprendiendo en dos *frentes* a la vez.

En este mes destacamos el artículo de Tratamiento de la imagen, con rutinas incluidas que permitirán a los lectores desarrollar efectos de retoque fotográfico en sus propios programas, para suavizar, enfocar y desenfocar imágenes, etc. El reconocimiento de voz es uno de los temas informáticos en el que se trabaja de forma especial en la actualidad, sus principios teóricos, así como un programa práctico, se incluyen en este número de SOLO PROGRAMADORES, para servir como base de una posterior investigación. En esa misma línea, las seis páginas de Sistemas Expertos son la primera aproximación a un tema en el que pretendemos reincidir a menudo: la posibilidad de que un ordenador adquiera, por sí sólo o con un tutor, sus propios conocimientos. Por último y dentro de los artículos puntuales: analizamos técnicas de Compresión Fractal. Todos ellos dan pie a investigar y experimentar con el PC.

Cambiando de tema, tal y como prometíamos el mes anterior, en este número publicamos los resultados del concurso de programación. Felicidades a los cinco ganadores y, en especial, al vencedor absoluto **Alvaro Begué Aguado**, de Valladolid, a quien deseamos se haya recuperado del *susto* que recibió cuando uno de los miembros de esta redacción le comunicó que había resultado ganador. Al resto de los participantes, nuestro agradecimiento. Esta primera edición ha estado plagada de anécdotas que se cuentan en el interior de la revista. Esperamos poderos ofrecer en breve otro reto en forma programa (se admiten ideas). Por cierto, hablando del concurso, una *mala* noticia para la comunidad masculina de programadores: si extrapolamos la proporción del número de programadores de cada sexo en función de los que han participado en este concurso la proporción es de 120 a 1 (o mejor dicho, a una). Esperemos que esta situación de *desequilibrio* cambie con el tiempo ¿verdad ?

Con esta pequeña broma, un mes más, llegamos al final deseando que la revista les guste a todos nuestros lectores. El próximo mes volveremos con nuevos temas y un programa sorpresa en el CD ROM. Gracias.

MARIO DE LUIS



MAYO 1995. Número 9
Es una publicación de
TOWER
COMMUNICATIONS, S.R.L.

Editor
Antonio M. Ferrer Abelló
Directora Comercial
Carmina Ferrer
Director de Producción
Carlos Peropadre
Publicidad
Magdalena Pedreño Llorente

••••••

Director
Mario de Luis García
Redactor Jefe
Carlos Doral Pérez
Coordinador técnico
Miguel Angel Alcalde

Colaboradores
María Gil, Juan Manuel Martín, Luis Martín, Rafael A. Cazorla, Agustín Guillén, Carlos Arias, Emilio Postigo, Enrique Coiras, Tomás Tejón, David Rubio, Sergio Rios, Julio Mateo, Juan José Samper, David Aparicio, Fernando de la Villa, Bernardo García, Ignacio Cea.

Edición técnica
Emilio Castellano
Maquetación
Fernando García Santamaría
Tratamiento de imagen
Josefa Fernández Martínez
Servicios Informáticos
Digital Dreams Multimedia, S.L.
Ilustraciones
Miguel Alcón
Secretaría de Redacción
Consuelo Jiménez
Suscripciones
Erika de la Riva
Redacción, Publicidad y Administración
C/ Marqués de Portugalete, 10
28027 MADRID
Telf.: 741 26 62 / Fax: 320 60 72
Filmación
Duvial
Impresión
G.D.B.
Distribución
MIDESA

Depósito legal: M-26827-1994
ISSN: 1134-4792

SUMARIO



SUMARIO





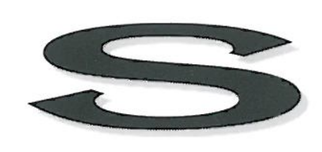






NOTICIAS





NUEVA VERSIÓN DE OPENDOC

La compañía Novell ha anunciado el lanzamiento de OpenDoc para Windows Developer Release 1 (DR1). Este producto incorpora la tecnología ComponentGlue de Novell y el System Object Model (SOM) de IBM, proporcionando interoperabilidad bidireccional con Object Linking and Embedding 2.0 (OLE) de Microsoft. De esta forma, Novell apuesta por el futuro de la mezcla y adaptación de los componentes del software. La tecnología ComponentGlue en el DR1 permite a los desarrolladores la creación de componentes OpenDoc que contengan objetos OLE1, OLE2 u OCX. De igual manera, los objetos OpenDoc actuarán como los objetos OLE u OCX cuando se incorporen en un contenedor OCX. También se permiten funciones de "arrastrar y soltar", portapapeles y vinculación de objetos OLE y componentes OpenDoc dentro y entre los documentos. OpenDoc para Windows DR1 incluye interoperabilidad con los servicios de script y automatización existentes. Es posible controlar componentes OpenDoc con cualquier controlador de automatización OLE, como Visual Basic. OpenDoc para Windows 95 y NT (DR2), incluirá capacidades adicionales de lenguaje script, estando su lanzamiento previsto para mayo. Por otra parte, la versión OpenDoc para Windows 3.1 estará disponible a principios del segundo trimestre del presente año.

Para más información:
Novell/WordPerfect España
Daniel Toledano
Tel: (91) 577 49 41

LA ARQUITECTURA BOOT BLOCK ADMITE PLUG AND PLAY

Intel ha anunciado que ya está disponible la memoria flash de 4 megabits para las aplicaciones de BIOS para máquinas PC. Esta memoria está basada en al arquitectura de alta integración Boot Block e incorpora la tecnología SmartVoltage.

Las BIOS Plug and Play necesitan una parte de memoria no volátil reescribible para almacenar 4 Kb de datos de configuración. La arquitectura Boot Block de Intel contiene dos bloques de parámetros adaptados a las necesidades de Plug and Play. El Boot Block asegura arranques fiables e impide el borrado o modificación accidentales del código de arranque durante actualizaciones de otras partes del circuito.

Las densidades de 2 y 4 megabits de la arquitectura Boot Block también proporcionan otras funciones BIOS como gestión de consumo, diagnósticos inteligentes, vídeo, SCSI, redes y los idiomas extranjeros.

La tecnología SmartVoltage a 3,3 voltios de Intel permite una mayor flexibilidad de programación y un modo de funcionamiento de bajo consumo. Para conseguirlo, admite la lectura a 3,3 voltios, la escritura a 12 voltios o una monotensión de 5 voltios por sencillez.

Para más información:
Intel Corporation Iberia
Zurbarán, 28 - 1º Izda.
28010 Madrid
Tel: (91) 308 25 52
Fax: (91) 310 54 60

SUSCRIPCIÓN A VISUAL C++

Microsoft Ibérica ha comenzado la comercialización de la primera versión del programa de suscripción a Microsoft Visual C++ y de las herramientas de programación en C++. La suscripción a Visual C++ se ofrece ahora como una actualización para los clientes de Visual C++ de España. Los suscriptores reciben tres veces al año

en CD-ROM las actualizaciones del sistema de desarrollo. Estas actualizaciones incluyen las nuevas características, mejoras de las MFC (Microsoft Foundation Classes), nuevo código de ejemplo y controladores, correcciones del compilador y artículos técnicos de interés para los desarrolladores. Además, incluyen las principales versiones de los productos cuando estén disponibles.

Entre las características que ofrece Visual C++ 2.1 se encuentran:

- MFC Migration Kit, herramienta que ayuda a los programadores a pasar el código C a las MFC.

- Nuevas prestaciones OLE Control Developers Kit (CDK), que permiten utilizar en los controles OLE las clases de bases de datos de las MFC, crear controles para Win32s y crear versiones internacionales por medio del soporte DBCS.

- Visual C++ 1.52, actualización de la versión 1.51, que incluye nuevas librerías de MFC de 16 bits con Simple MAPI, Windows Sockets y páginas de propiedad.

Para obtener la versión 2.1 hay que realizar la suscripción a Visual C++. Los usuarios de la versión 2.0 pueden actualizarlo por 21.150 pts (IVA y envío incluidos) y los de la versión 1.x pueden hacerlo por 51.113 pts (IVA y envío incluidos).

Para más información:

MicroSoft
Avda. Colmenar. Sector Foresta, 2, 3 4
Tres Cantos
28760 Madrid
Tel: (91) 804 00 00

NUEVA ESTACIÓN DE TRABAJO DE SILICON GRAPHICS

La compañía Silicon Graphics ha presentado Reality Station, la nueva estación de trabajo gráfica de un sólo procesador de la familia Onyx. Reality Station reduce el precio de entrada de los gráficos más rápidos del mundo en un 40 por ciento. Está destinada al mercado de diseño avanzado y fabricación, procesamiento de imágenes y simulación visual.

La estación de trabajo incorpora un microprocesador MIPS R4400 200 Mhz e incluye gráficos del subsistema gráfico de muy alto rendimiento Reality Engine2. Este subsistema está basado en una arquitectura de cálculo de crecimiento modular y cuenta con 1,2 GFLOPS de potencia de procesador de coma flotante.

Reality Station ofrece un conjunto de características gráficas tridimensionales en tiempo real, entre las que aparecen: topografía de texturas, "antialiasing" de polígonos, vectores y puntos, memoria intermedia para estereocopias interactivas de alta resolución, soporte de hardware para procesamiento de imágenes, salida de vídeo NTSC/PAL/S estándar y opciones de vídeo transmisión.

Por otro lado, Silicon Graphics ha anunciado una nueva opción Audio y Serie para el super-ordenador gráfico Onyx y la familia de servidores CHALLENGE. Esta tarjeta proporciona procesamiento de audio de calidad profesional y puertos serie de alta velocidad. Se utiliza a través de la biblioteca de audio de Silicon Graphics y permite trabajar con cuatro canales de audio de 48 KHz y 16 bits. El componente serie se compone de seis puertos serie de 115 Kbaud con baja latencia para dispositivos altamente interactivos.

Para más información:

Silicon Graphics, S. A.
Edificio "Santa Engracia 120"
Plaza del descubridor Diego de Ordás, número 3
28003 Madrid
Tel: (34-1) 442 90 77
Fax: (34-1) 442 01 50

DISCO MAGNETO-OPTICO DE 1,3 GB DE CAPACIDAD

Fujitsu España ha lanzado al mercado la unidad magneto-óptica IFD-1300-A de 1,3 Gb de capacidad en cada cartucho en formato de 5,25 pulgadas. Este dispositivo está pensado para aplicaciones que requieran alta integración de los datos y que manejan grandes cantidades de información como pueden ser archivos documentales, CAD/CAM, vídeo digital y multimedia.

Este disco magneto-óptico tiene un tiempo medio de acceso de 39 ms, una velocidad de transferencia de hasta 2 Mb/s, 300 revoluciones por minuto y una memoria de datos intermedia de 1 Mb. Se suministra montado en una caja externa junto con un cartucho de 1,3 Gb, para poder ser utilizado en el mínimo tiempo posible.

Los estándares que soporta son: el ISO-10089 para poder leer y escribir en cartuchos de 600 y 650 Mb, el ISO-3848/ para los cartuchos de 1,2 y 1,3 Gb, y el CCW para cartuchos WORM.

INTEL DOTA A UNA UNIVERSIDAD UN LABORATORIO COMPLETO

Intel ha puesto en marcha varios programas de apoyo para las universidades en Europa y otros países del mundo. Esta iniciativa constituye una inversión para el futuro por el apadrinamiento de universidades europeas. El primer donativo del programa paneuropeo es el dirigido a la Universidad irlandesa de Limerick, en forma de laboratorio de computación. Se trata de un laboratorio compuesto por 30 microordenadores basados en procesadores Pentium apoyados por 3 servidores de ficheros también basados en dicho procesador. Tiene un valor de unos 300.000 dólares y está dedicado a la enseñanza para el Colegio de Informática y de Electrónica.

Este programa de cooperación no está limitado a los establecimientos de enseñanza. Otros aspectos del programa se dedican a la formación tanto en cursos de estudios como en cursos de la vida profesional.

Intel Corporation Iberia
Zurbarán, 28 - 1º izda.
28010 Madrid
Tel: (91) 308 25 52
Fax: (91) 310 54 60

SYBASE E IBM INCREMENTAN EL PAPEL DEL MAINFRAME DENTRO DEL ENTORNO CLIENTE/SERVIDOR

El objetivo principal de esta iniciativa es facilitar que los mainframe puedan trabajar de forma cooperativa con aplicaciones cliente/servidor. Sybase planea desarrollar los API (Application Program Interface) Open Client y Open

Server para el sistema operativo MVS/Open Edition (OE) de IBM, la última versión del sistema operativo MVS de IBM para entornos mainframe. El acuerdo alcanzado entre ambas compañías permitirá el desarrollo adicional de productos de alta calidad diseñados para entornos cliente/servidor y mainframe.

Sybase también ha anunciado la disponibilidad del API Open Client para el sistema operativo MVS. Mediante este producto, miles de aplicaciones y TSO (Time Sharing Option) basadas en MVS podrán actuar como cliente en un entorno cliente/servidor para acceder directamente a una gran gama de fuentes de datos que no residen en una plataforma mainframe.

Por otro lado, Sybase e IBM han anunciado un acuerdo estratégico de desarrollo y marketing, diseñado para lanzar una serie de productos cliente/servidor líderes en la industria sobre el sistema operativo OS/2 para arquitecturas Intel y PowerPC.

SCO Y UNISYS DISTRIBUYEN SOLUCIONES CLIENTE/SERVIDOR

La corporación Unisys y Santa Cruz Operation, Inc. (SCO) han anunciado la firma de un acuerdo de marketing para la distribución de soluciones SCO para negocios críticos en entornos cliente/servidor de Unisys. Este acuerdo supondrá nuevos esfuerzos para ofrecer productos de alta tecnología, publicidad conjunta y cooperación de ambas compañías en potenciar las ventas y el desarrollo de aplicaciones.

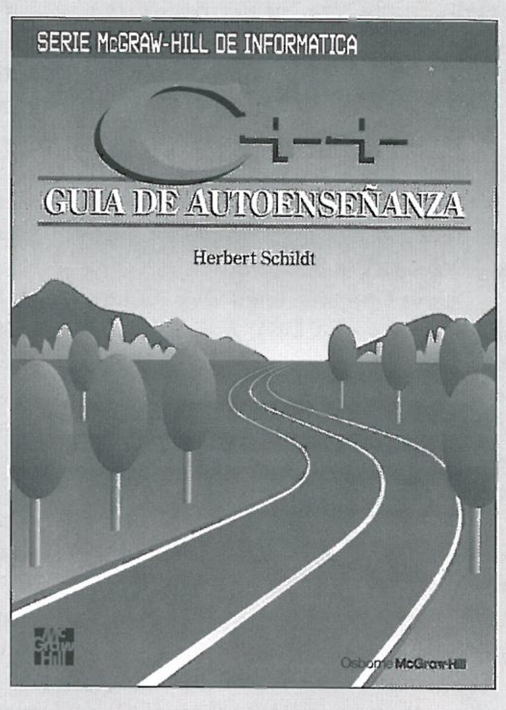
Como consecuencia del acuerdo, Lotus Notes y el sistema operativo SCO Open Server Enterprise serán preinstalados en los servidores PW2 de Unisys con procesadores Intel. Los servidores PW2 Serie Avanzada integrarán el software SCO Open Server, ofreciendo soluciones completas se sistemas UNIX para empresas pequeñas o de tamaño medio, redes y otros negocios.

Para más información:
Unisys España, S. A.
Avda. del Partenón, 4
Campo de las Naciones
28042 Madrid
Tel: (91) 721 12 12

LIBROS

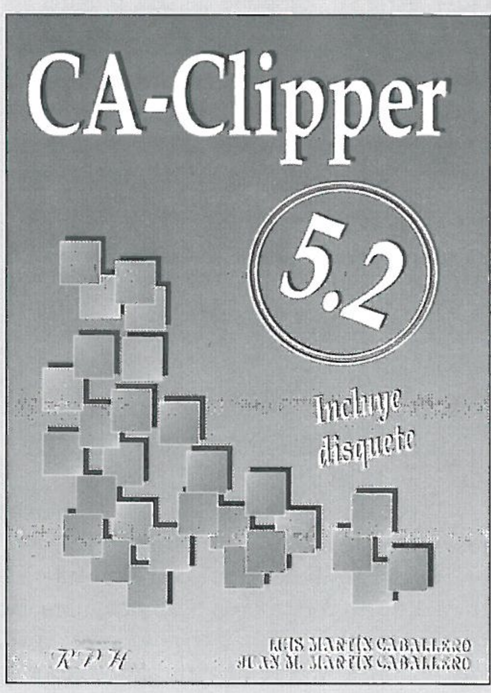
C++. GUÍA DE AUTOENSEÑANZA

Si tiene usted conocimientos en lenguaje C, esta obra pone a su alcance el lenguaje C++ y la programación orientada a objetos. Su autor, Herbert Schildt, demuestra una vez más su maestría con un libro muy didáctico, repleto de ejemplos, ejercicios y pruebas de aptitud para que el lector sea consciente en todo momento de su progreso. El libro emplea un método de aprendizaje supervisado, de modo que no se pasa al siguiente punto sin haber comprendido completamente el anterior. Para solucionar las posibles dudas, las últimas 140 páginas están dedicadas a dar respuesta a las pruebas y ejercicios propuestos.



Editorial: McGraw-Hill.
Autor: Herbert Schildt.
478 páginas.
Precio: 4000 IVA incluido.

CA-CLIPPER 5.2



Clipper se ha convertido en el lenguaje orientado a gestión de bases de datos más popular del momento. Con este libro se persigue que el lector llegue a ser capaz de desarrollar aplicaciones profesionales con la versión 5.2 del lenguaje. A través de sus páginas se hace un exhaustivo recorrido por todas las características de Clipper, junto con ejemplos y programas para facilitar el aprendizaje. También se tratan otros temas como la instalación del compilador, las guías Norton y las utilidades DBU y RL. Todos los programas de ejemplo se incluyen en un disquete que acompaña al libro.

Editorial: Publicaciones R. P. H.
Autores: Juan Manuel y Luis Martín Caballero.
650 páginas.
Precio: 5200 IVA incluido.





ENCONTRADO EN EL DISCO T:

En el gran centro de proceso de datos donde malgasto un tercio de mi vida (otro tercio lo malgasto durmiendo, y el otro, conduciendo un Ibiza entre mi casa y el gran centro de proceso de datos donde malgasto el primer tercio), el administrador de la red tiene la cortesía de mantener un gran disco virtual al que todo el mundo tiene acceso. El disco se borra todos los días a las seis de la mañana, y todo el personal del centro puede utilizarlo para almacenar ficheros temporalmente, o para intercambiar datos con personas de grupos de trabajo distintos al suyo, sin limitaciones de seguridad. Cuando copias algo al disco T:, pierdes instantáneamente toda esperanza de intimidad y permanencia. Eso sí, puedes pasar ratos divertidos hurgando en el disco y viendo cómo cambian cromos tus desconocidos compañeros de dos plantas más arriba (debo aclarar que éste es un centro grande, muy grande).

El disco T: es un efímero jardín donde todas las flores son de un día. El otro día vi allí un programa americano de dominio público que muestra "frases positivas" en pantalla a intervalos. Una especie de "fortune" de sesión continua que bombardea periódicamente al usuario con lindezas como: "Analice sus propios errores y concéntrese en mejorar". "Interésese auténticamente por los demás". "Respire profundamente para oxigenar sus células, e ingiera alimentos ricos en agua para limpiarlas". "Siéntese con una postura apropiada". No bromeo. Una especie de gran hermano asertivo y buenoide. De lo más eficaz para acabar vaciando el cargador de un M-16 en una hamburguesería llena de niños. Eso sí: se puede variar el fichero que utiliza el programa, para usar el conjunto de citas preferido. Se incluye el Evangelio de San Juan y selecciones del Libro de los Salmos. Edificante.

Pero otros de los ficheros de frases que se adjuntan tienen un tono menos solemne. Están las leyes completas de Murphy y gran número de corolarios, pensamientos cáusticos sobre la vida y la política, y sobre todo un ficherito donde se han concentrado frases de distinta procedencia sobre el mundo de la programación y la gestión de proyectos de software, y que me he permitido traducir al español para deleite del lector. Algunas de las frases son ya conocidas: otras se aplican a la vida en general, pero tienen una validez especial en este oficio nuestro. La mayor parte de las frases que se reproducen aquí pertenecen al libro "The Psychology of Computer Programming", de Gerald M. Weinberg. Esta misma noche mando un fax a la librería a ver si me consiguen ese libro. Tiene que ser atómico.

- Si se detecta que un programador es indispensable, lo mejor que se puede hacer es deshacerse de él tan pronto como sea posible.
- Hay programas que se deberían tirar a la basura antes de que lleguen a utilizarse.
- Poner a un montón de gente a trabajar sobre el mismo problema no les convierte en un equipo.
- Siendo como es la naturaleza de la programación, no existe relación entre el "tamaño" del error y los problemas que causa.
- Cuando un programador está pasando un mal rato buscando un error, es que está mirando en el sitio equivocado.
- La documentación es el aceite de ricino de la programación. Los jefes suponen que si tanto la odian los programadores, algo tendrá de bueno.
- Cualquier imbécil sin la capacidad de reírse de sí mismo y compartirlo, será incapaz de tolerar la programación durante mucho tiempo.
- El cerebro humano normalmente funciona al diez por ciento de su capacidad. El resto es sobrecarga del sistema operativo.
- La buena cocina lleva su tiempo. Si le hacemos esperar, es para servirle mejor, y para agradarle. - Menú del restaurante Antoine, Nueva Orleans
- La gestación de un niño tarda nueve meses, con independencia del número de mujeres asignado.
- Ley de Brooks: Meter más gente en un proyecto de software retrasado, lo retrasa más.
- Cuando se ha comprobado que todo funciona, y todo está bien integrado, aún quedan cuatro meses más de trabajo.
- Nunca te eches a la mar con dos cronómetros; llévate uno o tres.
- El problema era que todos los que trabajábamos allí, incluyéndome a mí mismo, queríamos hacer un buen trabajo, pero ELLOS no querían un buen trabajo, sólo querían mucho trabajo y rápido. - Rick Baker, técnico de maquillaje de King Kong, La Guerra de las Galaxias, y otros.
- La generación de números aleatorios es demasiado importante para abandonarla al azar.
- No conozco ninguna razón por la que no pudiéramos hacerlo, pero quizás podamos pensar en una.
- El usuario no sabe lo que quiere hasta que ve lo que le dan.
- El himno nacional del programador es: "AAAAAAAGGGGGGGGGHHHHHHH".

Para no ser acusado de escribir columnas íntegramente con el material de otros, contribuiré con dos frases de mi cosecha:

- Lo que se diseñó como provisional resultó ser permanente. Lo que se diseñó como permanente resultó ser provisional.
- Ley de la fatalidad de las empresas de servicios: Si no sabes si es peor un jefe o un cliente, es que nunca te han puesto a un cliente de jefe.

Y por último, mi favorita:

NOTICIA DE ULTIMA HORA: Programador encontrado muerto en la ducha agarrando una botella de champú que decía: lavar, aclarar, repetir.





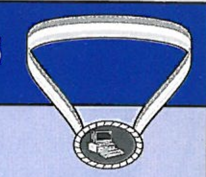
4 en Raya

Ya tenemos los ganadores del primer torneo de 4 en raya por ordenador organizado por Sólo Programadores. De entre los 130 participantes 5 de vosotros os vais a llevar a casa la última versión del estupendo compilador de Borland C++.
Si quieres saber quiénes son los ganadores sigue leyendo y no te pierdas ni un detalle de lo sucedido.
Como datos curiosos a reseñar en la competición cabe destacar la poca participación femenina. De los 130 participantes, María Teresa Carbonell se ha quedado "Sola ante el peligro" y ha sido la única chica que ha participado en el torneo.
También podemos mencionar algo con respecto a los lenguajes de programación que no extrañará a nadie. Debido a su gran difusión el lenguaje C ha sido el rey de este torneo, seguido a distancia del Pascal, del C++, cada vez se ven más aplicaciones desarrolladas usando técnicas de programación orientada a objetos, y del ensamblador. Pero no son éstos los únicos lenguajes elegidos por nuestros participantes. Con una menor participación, pero no por ello menos importante, hemos recibido aplicaciones desarrolladas en BASIC (QBASIC y Turbo BASIC), en Modula-2 y en Clipper 5.2.
Hay provincias cuya participación ha sido nula. Estamos convencidos de que en ellas hay programadores de una calidad excepcional. Que no se diga que no hay buenos programadores en "toda" España.
En el próximo mes se incluirán en el disco que acompaña a la revista, todos los ficheros que reflejan el desarrollo de las partidas así como los cinco programas ganadores y una versión del Árbitro.

Los preliminares

Una vez reunidos todos los participantes se procedió a su verificación individual. El chequeo de virus permitió descubrir que dos de los programas enviados estaban infectados por el virus Barrotes y el Whisper respectivamente. Afortunadamente el programa detector fue capaz de eliminarlos sin problema.
A continuación se jugó una ronda preliminar en la que todos los participantes jugaron una partida contra el programa "Aleator" (el programa de ejemplo que acompaña a las bases del concurso y que realiza jugadas de forma aleatoria). Con ello no sólo se verificaba el correcto funcionamiento de los programas enviados al concurso, sino del programa ARBITRO encargado de controlar los combates. Como dato anecdótico cabe reseñar el hecho de que el jugador aleatorio ganó una de las partidas.

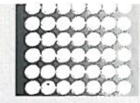
GANADORES



PROGRAMA:	NOMBRE:
1º 0000125.exe	**Alvaro Begué Aguado (Valladolid)**
2º 00000030.exe	Juan Ignacio Romera Arroyo (Madrid)
3º 00000082.exe	Miguel Angel Estremera (Barcelona)
4º 00000027.exe	Gaizka Solano Moral (Vitoria)
5º 00000129.exe	Climent Puig Ballus (Gironella)

Desarrollo del concurso

Una vez chequeados los programas participantes se procedió a su división en cinco grupos de 26 participantes cada uno. Este reparto se realizó de forma aleatoria por medio de un programa -GENGRUP.EXE- creado para tal fin. Los archivos GxF1.NOM, donde x es un número de 1 a 5, contienen las listas de los programas incluidos en cada uno de los grupos.
Antes de continuar es necesario explicar la notación utilizada en los nombres de los archivos de resultados. Como ya hemos visto existen cinco grupos. Cada uno de estos grupos va compitiendo por parejas, eliminándose la mitad de ellos en cada bloque de combates. A cada uno de estos bloques se le denomina fase y tienen el mismo significado que en cualquier competición deportiva. Ejemplos de fase serían Cuartos de Final, Semifinales, etc. Por tanto un grupo n va realizando fases desde GnF1 hasta que sólo quede un único participante: el finalista.
Así pues el desarrollo del concurso está controlado por dos programas: -GENFASE.EXE- que se encarga, a partir del fichero GxFy.NOM de generar las parejas de programas que se enfrentarán de forma aleatoria. Si en una fase hay un número impar de participantes quedará un jugador desemparejado. A este jugador se le asignará como contrincante un "jugador de relleno", que no es más que otro participante de su mismo grupo. Si gana el jugador desemparejado pasará a la siguiente fase.
Al jugador de relleno no se le será tenido en cuenta este combate.
El programa -ARBITRO.EXE- va realizando los combates y genera dos tipos de ficheros:
- GxFy.LOG: Es una lista de los encuentros jugados en cada fase así como su resultado final. El formato es el siguiente:



Fecha Hora: número de combate: Participantes: Partidas jugadas Resultado final

Ejemplo:
26/03/1995 11:23:29.25: 2: 01010101-1111111: Partidas=2 Marcador 0-2

- GxFyCz.LOG: Lista todos los acontecimientos producidos durante un combate entre dos programas: movimientos, tiempo, causas de descalificación, resultados parciales y resultados finales. Su formato es el siguiente:

Fecha Hora: (número de turno) jugador y movimiento Tiempo=tiempo empleado en la jugada.
Ejemplo:
26/03/1995 11:32:11.05: (33)Jugador 1 mueve en columna 3 Tiempo=10.93 seg.

Las líneas de mensaje tienen el siguiente formato:
Fecha Hora: Mensaje
Ejemplo:
26/03/1995 11:32:11.10: —> Resultado parcial:
00000001=1
00000011=1
GxFyCz.DES: Contiene el estado del tablero en el momento en que se produjo una descalificación.
Como es previsible Arbitro se encarga también de verificar que ambos participantes cumplen las bases del concurso y finalizar la partida cuando se cumple uno de los tres motivos para ello: Cuatro en raya, Tablas o Descalificación de uno de los contendientes.
Las posibles causas de eliminación del torneo son las siguientes:

1. Comprobaciones sobre el fichero TABLERO.DAT
   a) El fichero debe existir y tener la longitud adecuada (42 bytes).
   b) Se compara el tablero nuevo con el anterior. Únicamente puede haber variado una posición, que anteriormente valía 0 y ahora contiene el número del jugador que ha movido.
   c) Debajo de la fichas movidas no puede haber casillas vacías.
2. Comprobaciones adicionales:
   a) Tiempo: Un jugador no puede emplear más de 12 segundos por jugada. Se ha añadido un margen de 3 segundos (un 25%) de error, con lo que un programa es descalificado si emplea más de 15 segundos en alguna jugada.

   b) Memoria: Si algún programa deja la memoria en mal estado o sin liberar es descalificado.
   c) Fichero de datos: No puede superar el tamaño indicado en las bases del concurso.

Cada combate entre dos programas se compone de dos partidas en las que comienza alternativamente cada uno de ellos. Una partida ganada supone un punto para el ganador y en caso de empate ninguno de los jugadores recibe punto. Al final de las dos partidas se comprueba el marcador. Si hay empate se mira el jugador que ganó en menor número de movimientos y se le da un punto de desempate. Si ambas partidas fueron tablas o si los dos jugadores ganaron en el mismo número de movimientos se continua jugando partidas hasta que se rompa el empate.
En la liguilla final se aplicó la misma regla, pero permitiendo el empate. Al final de cada combate (dos partidas) se le daban 2 puntos al jugador que ganaba o uno a cada uno en caso de empate. La suma de los puntos daba el resultado final.

Resultado de la liguilla entre los cinco finalistas

Programa	Jugadas	Ganadas	Empate	Perdidas	Total	Puesto
125	*4*	*4*	*0*	*0*	*8*	*1*
030	4	3	0	1	6	2
081	4	0	3	1	3	3
129	4	0	2	2	2	4
027	4	0	2	2	2	4

Resultados

Los cinco finalistas han sido las siguientes personas:
Grupo 1: Gaizka Solano Moral (Vitoria)
Grupo 2: Alvaro Begué Aguado (Valladolid)
Grupo 3: Miguel Angel Estremera Paños (Barcelona)
Grupo 4: Climent Puig Ballús (Gironella-Barcelona)
Grupo 5: Juan Ignacio Romera Arroyo (Madrid)
Los ficheros con los resultados detallados de cada partida de la liguilla final se encuentran en los archivos G0F0Cx.LOG, donde x es un número de 1 a 10.

El programa 4 en Raya

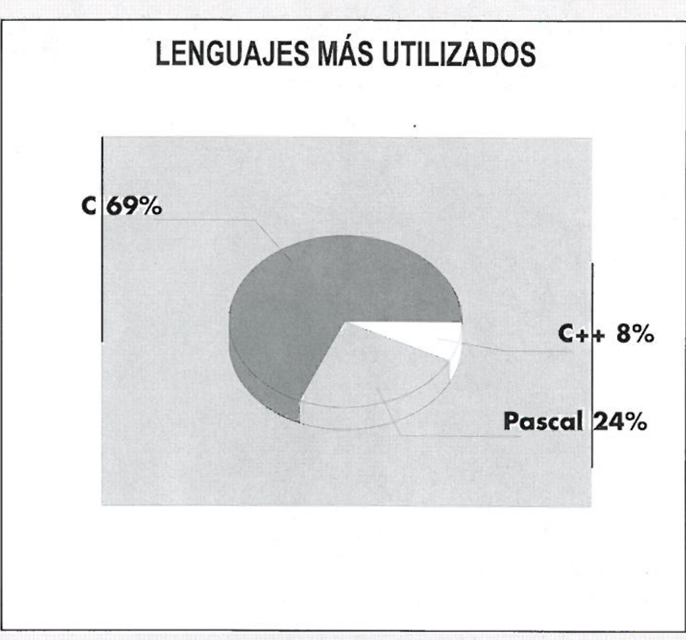
En el disco de este mes se encuentra una versión del programa ARBITRO que se llama 4RAYA.EXE, dentro del directorio CONCURSO. Este programa permite jugar a una persona contra cualquiera de los 130 programas enviados al concurso, a dos programas entre sí o a dos personas. El programa es bastante simple y basta con seguir las instrucciones que aparecen en pantalla.

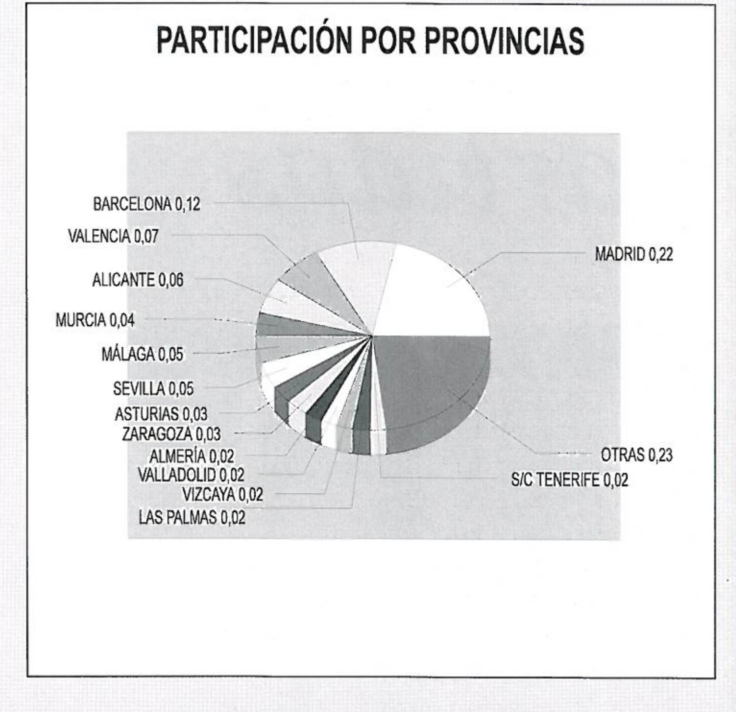
LISTA DE LOS JUGADORES

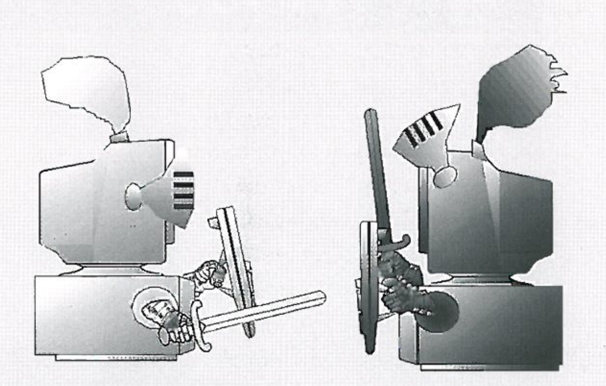
001 José Luis Nates Carranza (Madrid)
002 Jesús Javier Montero Gómez (Leganés)
003 Ruben Sánchez Peña (Madrid)
004 Joaquin Herreros Trigueros (Madrid)
005 Eugenio de la Torre (Madrid)
006 Juan Antonio Clavero (Madrid)
007 Alberto Juan Gómez (Madrid)
008 Jose Antonio Solbes (Madrid)
009 Antonio M. Terriza (Ciudad Real)
010 Juan Galán Leal (Manzanares)
011 Daniel Rodriguez Pulpillo (Madrid)
012 Manuel Fournier Martínez (Madrid)
013 Gonzalo León Manzano (Albacete)
014 José Antonio Cabrera Jiménez (Almería)
015 Alberto Gómez Vicente (Oviedo)
016 Jesús María Chamizo (Fuenlabrada)
017 Demetrio Fdez. Alvarez (Oviedo)
018 Marcos Redondo Fonseca (Avilés)
019 José Escuadra Burrieza (Zamora)
020 Miguel Angel Virto García (Madrid)
021 Tomás Pastor Báez (Valladolid)
022 Raúl Izquierdo Castanedo (Santander)
023 Daniel Pérez Cabellos (Santander)
024 Jesús María Calleja (Villafranca)
025 Lorenzo Eizmendi Apellaniz (Pamplona)
026 Rafael Guereta Nazabal (Vitoria)
027 Gaizka Solano Moral (Vitoria)
028 Fco. Javier Martinez de Pison (Logroño)
029 Pedro F. Maicas Alijarde (Zaragoza)
030 Juan Ignacio Romera Arroyo (Madrid)
031 Javier Arnal Sastre (Zaragoza)
032 Miguel Angel Ferrer (Zaragoza)
033 Carmelo Jerónimo Morales (El Ejido)
034 Salvador Ferris (Algemesi)
035 Joaquín Manzano Sempere (Santa Pola)
036 Bernardo Miguel Iglesias (Alicante)
037 Juan Antonio Gil (Alicante)
038 Ramón Bernat García (Valencia)
039 María Teresa Carbonell (Alginet)
040 Alfonso Javier Acosta (Cartagena)
041 José Manuel Bocanegra Cruz (Granada)
042 Rafael Gutierrez Martinez (Valencia)
043 Manuel Ciges Faura (Benidorm)
044 José Luis Triviño (Málaga)
045 Miguel Antonio Ballesteros (Málaga)
046 Pedro Fernández Presa (Granada)
047 Rafael Carlos Caballos (Sevilla)
048 Francisco Rueda Alvarez (Marbella)
049 Moises José del Amor (Murcia)
050 Antonio Manuel Rodriguez (Murcia)
051 Pablo José López (Murcia)
052 Juan Manuel Martínez Pastor (Albox)
053 Juan Manuel Cintado (Sevilla)
054 Angel Pérez Reina (Sevilla)
055 Rafael Angel Eslava (Sevilla)
056 José Luis Pérez Barrales (Sevilla)
057 Juan Desongles Corrales (Sevilla)
058 Luis Miguel Lacosta Jiménez (Zaragoza)
059 Iñigo Uriondo Urquijo (Llodio)
060 Juan Carlos Madrazo (Bilbao)
061 Sergio Lizundia López (Bilbao)
062 Javier Gómez de Cos (Cádiz)
063 Juan José Berlanga (Algeciras)
064 Francisco otero García (La Coruña)
065 Miguel Angel Pereira (La Coruña)
066 Narciso Luque Carrillo (Málaga)
067 Manuel Toscano Moreno (Málaga)
068 Carlos Mira Gil (Málaga)
069 Manuel Juni Marín (Sabadell)
070 Fco.Javier Martinez Arpa (Castelldefels)
071 Florentino Avila (Lasarte-Oria)
072 Juan Serra Ferrando (Barcelona)
073 Vicente J. Sánchez (Murcia)
074 Gerardo Acevedo Rua (Vigo)
075 Gabriel Martí i Fuentes (Calella)
076 Julio Camacho Pont (Barcelona)
077 Pedro Albarracín (L'Hospitalet de LL)
078 Pere Viladoms Serra (Barcelona)
079 David Portabella Clotet (Manresa)
080 Juan Javier Poy (Tarragona)
081 Jesús Valladolid Rebollar (Roses)
082 Miguel Angel Estremera (Barcelona)
083 Juan Carlos Rodríguez (Barcelona)
084 Arturo Jose Esparragón (Las Palmas GC)
085 Aitor Vázquez Larrea (Palma de Mallorca)
086 José Luis Calvo Salanova (Madrid)
087 Gines Vidal Bravo. (Palma Mallorca)
088 Juan Martínez Miranda (Barcelona)
089 Joaquín Font Juvanteny (Olot)
090 Sergio Alonso Robles (Barcelona)
091 Antonio Javier Guerra (Arucas)
092 José Manuel Rodríguez (S.C. de Tenerife)
093 David Abilleira Freijeiro (Pontevedra)
094 Pedro Moreno Simón (Córdoba)
095 Andrés del Campo (Córdoba)
096 Saulo Alvarado Mateos (Las Palmas GC)
097 Carlos Navarro (Petrel)
098 José Escandell Sánchez (Elda)
099 José Luis Santos Gómez (Valladolid)
100 José maría Méndez Ruiz (Melilla)
101 David Vargas Ruiz (La Laguna)
102 Victor Rodriguez Herreros (La Laguna)
103 Abel Manuel Bernabeu (Callosa de Segura)
104 Emilio Jesús Sarmiento (Badalona)



105 Pedro Lumbreras Gili (Igualada)
106 José María López Antequera (Andorra)
107 José Ignacio de Miguel (Alicante)
108 Alfredo Hernández (Madrid)
109 Jesús Angel de Gregorio (Madrid)
110 Oscar José Pascual Marcos (Madrid)
111 Pablo de la Flor (Madrid)
112 José Alberto Aparicio (Madrid)
113 Alfonso García-Patiño (Madrid)
114 Alberto Iglesias Fermosell (Madrid)
115 Tomás Arribas Navarro (Madrid)
116 Antonio Presa Vázquez (Madrid)
117 Jesús Jiménez Parra (Madrid)
118 César María Vicente (Añover de Tajo)
119 Fco. Javier García Cenamor (Getafe)
120 José Luis San Emeterio (Covielles)
121 Jesús Angel de Miguel (Alicante)
122 David Blanco Serrano (Torrejón de Ardoz)
123 Farid Fleifel Tapia (Mislata)
124 Luis Pablo Nieto Centeno (Madrid)
125 Alvaro Begué Aguado (Valladolid)
126 Juan José Eraso Escalona (Calahorra)
127 Fco. Javier Blanes Pla (Valencia)
128 Josep Oriol Pique (Terrassa)
129 Climent Puig Ballus (Gironella)
130 Leonardo José Zayas Vila (Salamanca)

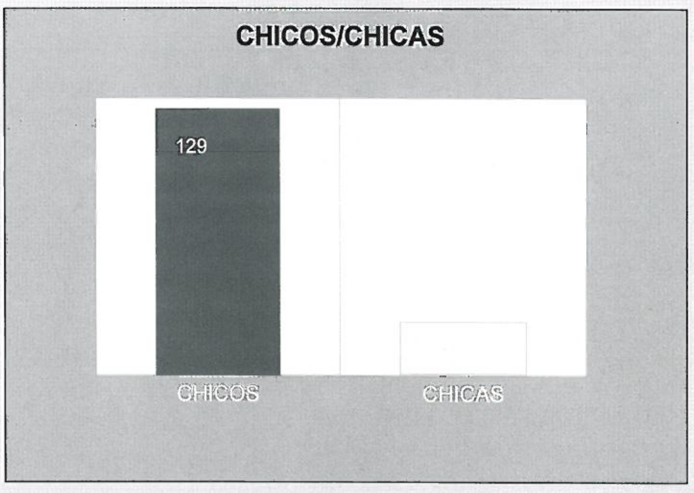




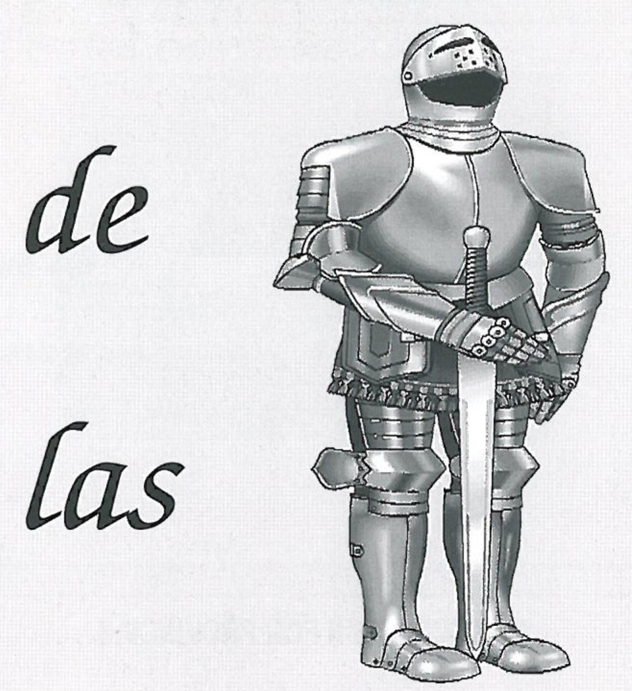


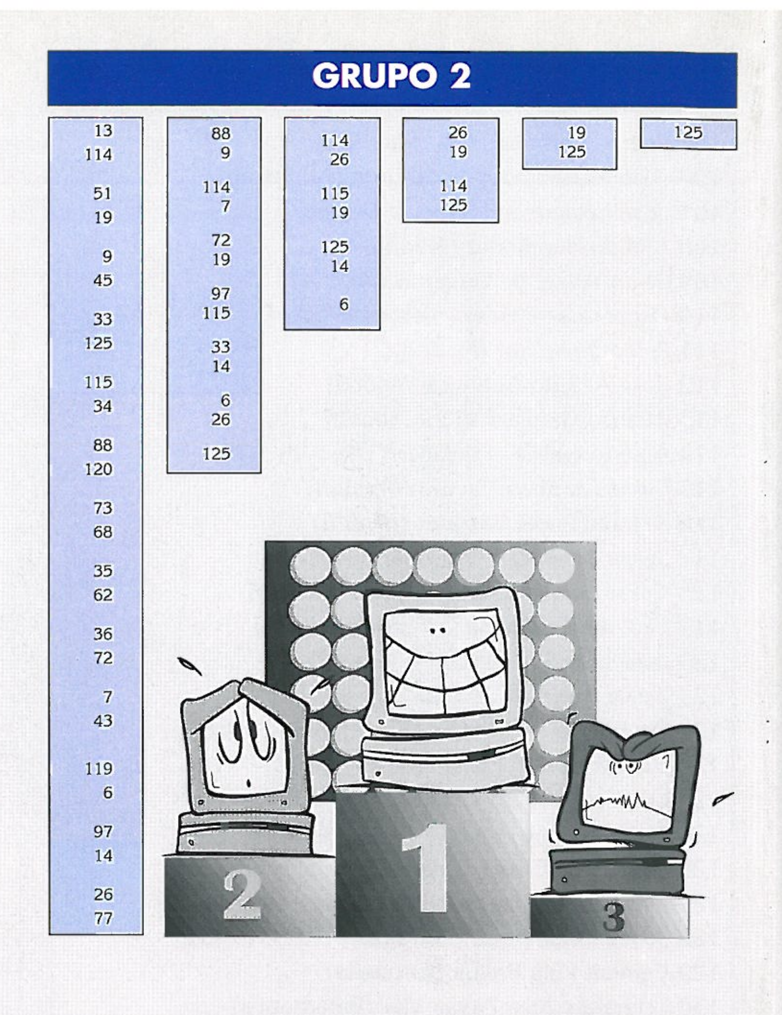
LOS 10 PROGRAMAS MAS LARGOS

F. Javier Martinez	Tomás Pastor
Javier Arenal	Angel Perez
Juan Javier Poy	Demetrio Fernández
Jose I. de Miguel	Joaquín Herreros
José Escandell	Narciso Luque



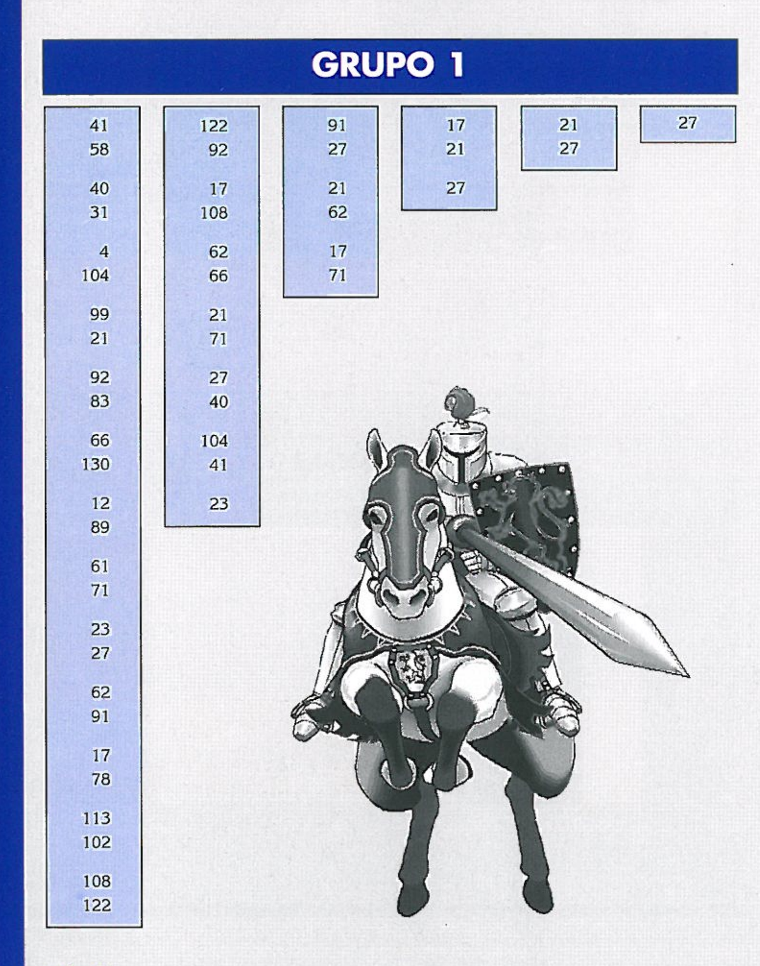
Desarrollo de las partidas





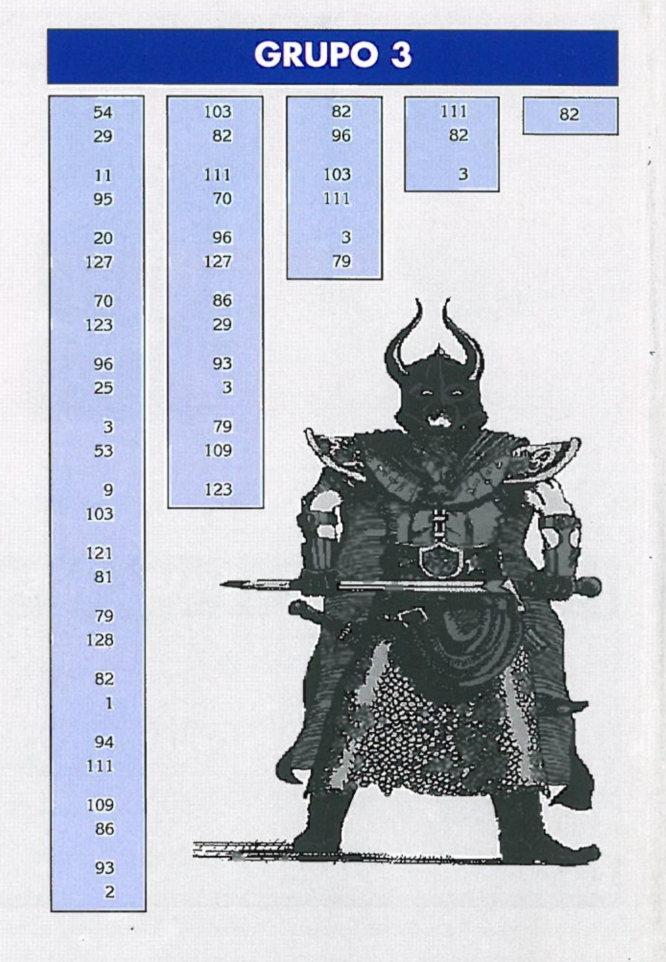
GRUPO 2

13	88	114	26	19	125
114	9	26	19	125	
51	114	115	114		
19	7	19	125		
9	72	125			
45	19	14			
33	97	6			
125	115				
115	33				
34	14				
88	6				
120	26				
73	125				
68					
35					
62					
36					
72					
7					
43					
119					
6					
97					
14					
26					
77					



GRUPO 1

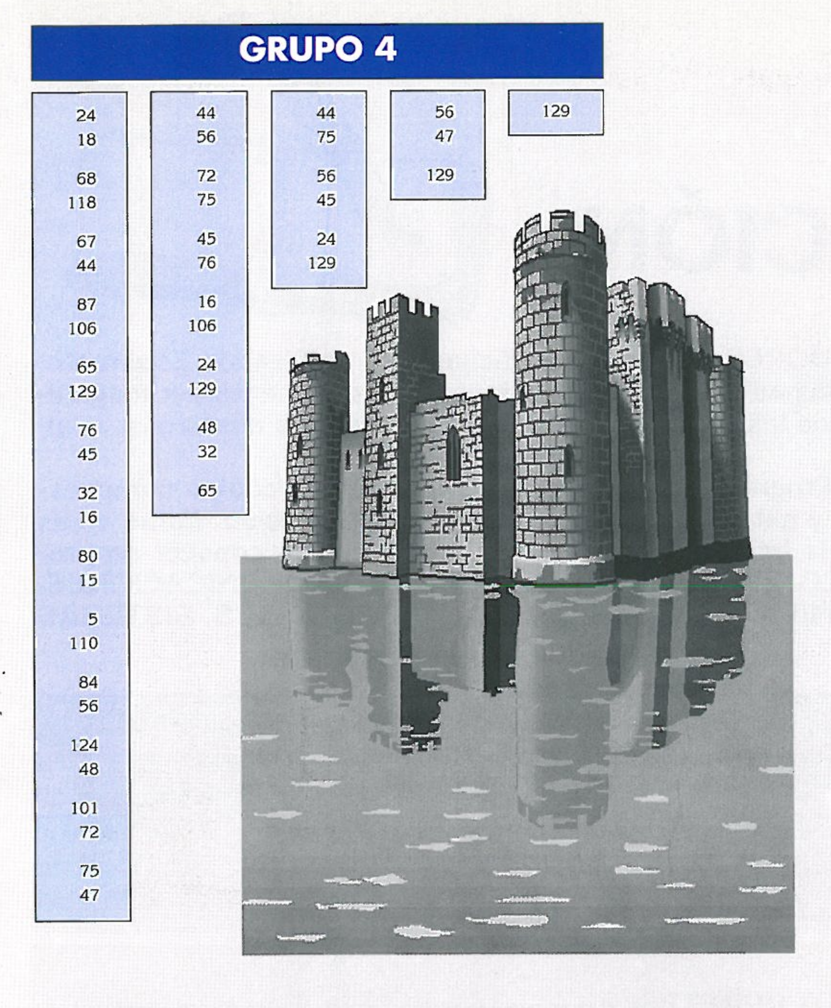
41	122	91	17	21	27
58	92	27	21	27	
40	17	21	27		
31	108	62			
4	62	17			
104	66	71			
99	21				
21	71				
92	27				
83	40				
66	104				
130	41				
12	23				
89					
61					
71					
23					
27					
62					
91					
17					
78					
113					
102					
108					
122					



GRUPO 3

54	103	82	111	82
29	82	96	82	
11	111	103	3	
95	70	111		
20	96	3		
127	127	79		
70	86			
123	29			
96	93			
25	3			
3	79			
53	109			
9	123			
103				
121				
81				
79				
128				
82				
1				
94				
111				
109				
86				
93				
2				

GRUPO 4

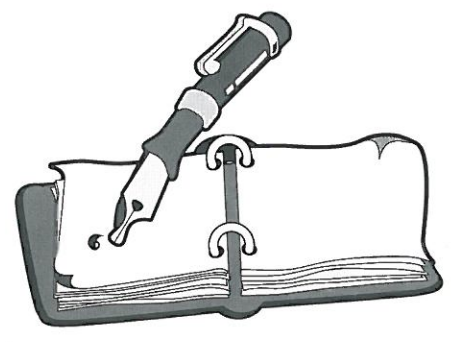


24	44	44	56	129
18	56	75	47	
68	72	56	129	
118	75	45		
67	45	24		
44	76	129		
87	16			
106	106			
65	24			
129	129			
76	48			
45	32			
32	65			
16				
80				
15				
5				
110				
84				
56				
124				
48				
101				
72				
75				
47				

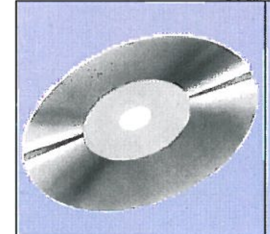
GRUPO 5

30	30	38	28	28	30
49	38	64	30	30	
55	85	30	60		
126	60	85	64		
85	107	28			
90	28	60			
38	64	112			
60	50				
28	37				
51	8				
60	112				
100	117				
56	60				
98					
8					
50					
105					
112					
116					
64					
117					
39					
42					
107					
22					
37					

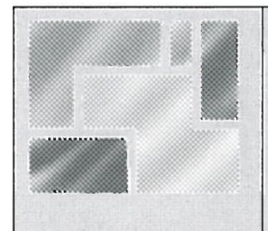
FICHAS DE PROGRAMACIÓN



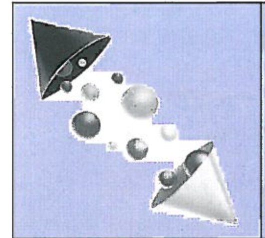
El pasado número 8 de SÓLO PROGRAMADORES contenía unas fichas y un archivador. Este material compondrá junto con todas las demás fichas que se van a entregar periódicamente, un magnífico archivo documental donde encontrar todos los datos y recursos necesarios para desarrollar cualquier tipo de aplicación.
Los 10 temas principales de las Fichas de Programación han sido elegidos por un equipo compuesto por colaboradores de la revista, lectores y naturalmente por el equipo de Redacción. Estas áreas pretenden englobar todos los aspectos de la Informática profesional que necesita conocer un programador, y son los siguientes: CD-ROM, COMPILADORES E INTÉRPRETES, COMUNICACIONES, ENTRETENIMIENTO, HARDWARE, HERRAMIENTAS, LENGUAJES Y METODOLOGíAS, SISTEMAS OPERATIVOS, TRUCOS y VARIOS.



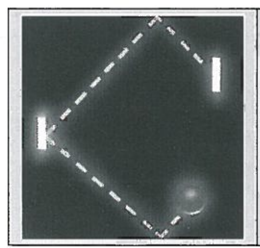
CD-ROM: En el mercado existen en este formato centenerares de librerias, herramientas y utilidades específicas para programadores, así como gran cantidad de shareware. En este área pueden encontrarse CD-ROMs comentados y analizados para poder elegir cual es el más idoneo cuando surge una determinada necesidad.



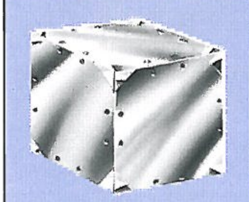
COMPILADORES E INTÉRPRETES: Éstos son los útiles de trabajo de los programadores y es fundamental conocerlos perfectamente. Teclas de acceso rápido, opciones de compilación, características generales, ventajas e inconvenientes y funciones avanzadas, son algunas de las fichas que se pueden encontrar en este apartado.



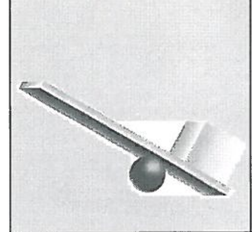
COMUNICACIONES: En un mundo donde la información es el principal recurso, hay que dominar todas las técnicas de acceso a los grandes núcleos donde reside, así como saber aprovechar estos medios para crear aplicaciones más potentes. Internet, modem, BBS, redes locales, RDSI, son los temas tratados en esta sección, tanto de forma teórica como práctica.



ENTRETENIMIENTO: El trabajo de un programador es a veces obsesionante, horas y horas pensando en cómo resolver ese misterioso "bug". Un buen juego, chiste o acertijo es capaz de evitar ese colapso nervioso, que parece abatirse sobre el informático cuando descubre que el bug se debía a que la impresora no tenía papel.



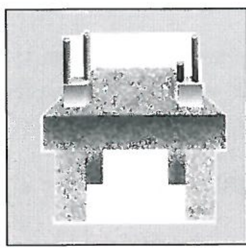
HARDWARE: Desde el comienzo de la informática, los programadores han trabajado con máquinas compuestas por centenares de dispositivos. Estos aparatos afectan directamente al funcionamiento de las aplicaciones, por lo que saber cómo funcionan, cómo se programan y qué características poseen frente a otros similares, es algo que todo informático debe conocer.



HERRAMIENTAS: Esta sección está dedicada a todos los productos que completan el conjunto de herramientas que puede necesitar un programador. Editores de texto, formateadores de código, librerías, lenguajes de autor, e incluso herramientas CASE y gestores de proyectos son los contenidos de estas fichas.



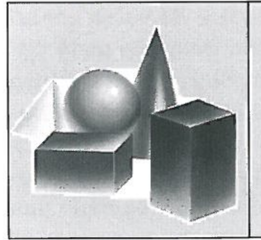
LENGUAJES Y METODOLOGÍAS: Lenguajes como Pascal, Lisp, C, Ada, Prolog, Modula-2, C, Basic, Ensamblador, Fortran, Clipper, Oracle, Adabas, Natural, ... y metodologías de diseño como Warnier, Bertini, Jackson, Orientación a objetos, tienen que estar al alcance de la mano de cualquier programador para no quedarse rezagado tecnológicamente.



SISTEMAS OPERATIVOS: Este es el auténtico motor de los ordenadores y es muy importante conocer como usuario y como programador todos los detalles que éstos poseen. En esta sección se podran encontrar fichas con trucos, análisis de las últimas versiones y programación específica de S. O. como DOS, MS-Windows, OS/2, UNIX o MAC System.



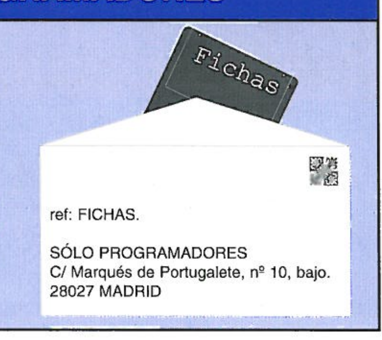
TRUCOS: Gran parte de la programación se basa en pequeñas ideas que de una forma genial solucionan el mayor problema que se pueda imaginar. Aquí se podrán encontrar desde los más clásicos algoritmos de ordenación hasta el método más veloz para animar una secuencia de imágenes, así como los más diversos formatos de ficheros de datos.



VARIOS: Un ordenador es la máquina más imprevisible que pueda existir, a pesar de presumirse como todo lo contrario, por esto se ha incluido este apartado. Lo que puede contener es un misterio. Quizás se encuentre una explicación para crear vida artificial dentro del ordenador, o cómo construir un robot a partir de aparatos caseros, ...

PARTICIPA CON TUS IDEAS EN LAS FICHAS PARA PROGRAMADORES

Sólo Programadores te ofrece la posibilidad de publicar firmado tu mejor truco, algoritmo, ayuda a la programación, comentario sobre alguna herramienta, técnicas específicas de programación, etc en las Fichas para Programadores.
Además, Sólo Programadores sorteará, entre todas las fichas recibidas (no sólo las publicadas), cinco fantásticos juegos de ordenador.
Las fichas recibidas deberán constar de dos páginas tecleadas en un editor de textos como Ms-Word o Word Perfect (formato DIN A-4), más las ilustraciones a mano alzada y capturas pertinentes en formato TIF, PCX, ... Conviene enviar los disquetes protegidos para evitar su deterioro durante el viaje. Esperamos contar pronto con tus creaciones.



ENTREVISTA

CHARMED

Charmed S.L. es la empresa creadora de "Amazonia La tierra de las aguas", el primer documental español en CD-ROM, que fue lanzado al mercado el pasado mes de Enero. Este CD-ROM interactivo propone un recorrido audiovisual por la mayor selva tropical del planeta.

Charmed S.L. es la empresa creadora de "Amazonia La tierra de las aguas", el primer documental español en CD-ROM, que fue lanzado al mercado el pasado mes de Enero. Este CD-ROM interactivo propone un recorrido audiovisual por la mayor selva tropical del planeta.

Este proyecto se realizó durante varios viajes a la selva amazónica entre 1993 y 1994, contando con el asesoramiento de importantes instituciones brasileñas, como es la FCBA (Fundaçao para Conservaçao da Biodiversidade da Amazonia, Fundación para la Conservación de la Bio-Diversidad).

Este recorrido por la selva tiene una duración de entre 12 y 20 minutos de imágenes, con locución e informaciones suplementarias, además de los vídeos que ilustran las diferentes regiones del Amazonas.

Se tarda aproximadamente una hora en hacer un recorrido por el programa, en la que el espectador deja de ser un sujeto pasivo para interactuar con el documental.



¿A qué se dedica Charmed?

Charmed nace hace 5 años aproximadamente, y lo hizo gracias a la aportación de gente que ya había trabajado en el sector de la imagen, tanto fotográfica como de vídeo, y surge para ofrecer una serie de servicios a determinadas empresas, tanto de Marketing como de formación.

Lo último que hemos realizado han sido trabajos de programación desde hace casi dos años y medio. En principio estaban muy ligados al tema de la imagen, para hacer o bien animaciones en 3D o rotulación para imagen, y posteriormente hemos entrado de lleno en la programación multimedia.

¿Cuántas personas componen la empresa?

Somos cuatro personas trabajando en la empresa, más algunas otras que nos ayudan y colaboran en determinados proyectos.

El equipo que dirige Charmed es el siguiente:

Joaquín Sánchez. Gerente, estudió Sociología y se especializó en el campo de la fotografía y del vídeo.

Javier Sánchez. Responsable del área de Informática, estudió en la E.U. de Informática de la U.P.M.

José Antonio García. Relaciones Públicas y reportero gráfico.

¿Con qué medios disponéis para trabajar?

Para hacer trabajos de vídeo contamos con un equipo Betacam tanto para grabación como edición, y también con algunos medios informáticos basados en PC, con hardware específico para programación multimedia y retoque fotográfico, como por ejemplo una tarjeta gráfica Targa.

¿Cuál ha sido el proyecto más importante de programación que halláis realizado hasta el momento?

El programa más importante que hallamos realizado de programación ha sido "Amazonia: La tierra de las aguas", que surgió por un esfuerzo común entre José Antonio y nosotros, ya que teníamos mucho interés en realizar este tipo de trabajos, pensando que tenían un atractivo desde el punto de vista visual muy importante, tanto pa-

ra realizar algunos documentales televisivos, como también para realizar algunos programas de tipo multimedia. La idea en principio era realizar una serie documental para televisión, que seguramente saldrá, pero se ha impuesto el CD-ROM.

Las dificultades de preparación de un trabajo de este tipo empiezan no sólo por los puramente informáticos, sino con la preparación de verdaderas expediciones.

Llegamos a la ciudad de Manaos, la capital del estado del Amazonas, y desde allí hemos realizado viajes en todos los medios de transporte imaginables .

En cuanto a los datos que ofrecemos en el CD-ROM, tienen validez porque están respaldados por esta fundación así como por los conocimientos que posee José Antonio García, al llevar más de siete años viajando por la zona, y la conoce bastante bien.

La verdad es que fue una experiencia muy bonita, y mereció la pena pues significaba introducir en la empresa una producción propia, la cual creíamos que sería estupendo llevar a la gente. La conjugamos con nuestras ganas de aventura y de hacer con nuestra vida algo que se sale un poco de los límites normales, de los viajes que puede realizar cualquiera.

Ya de vuelta en Madrid, pensamos que en España nunca se habían hecho documentales en CD-ROM, y nos pareció que era un vehículo válido, como ha demostrado ser, con mucho futuro, y a partir de ahí Javier comenzó a trabajar en el programa. Creo que hemos conseguido un producto muy interesante, y de hecho en el mercado así nos está respondiendo.

La calidad de Amazonia se debe a que las grabaciones están realizadas en Betacam, que es un formato de televisión con unas cualidades muy buenas, por lo que hemos querido garantizar la perfecta visualización de las imágenes, mediante un cuidado proceso en la manipulación de las imágenes, tanto en la compresión como en la digitalización y retoque de estas. Todo ello se ha hecho con el mayor esmero, y con la mayor delicadeza que hemos podido. Para el estado actual de la técnica, y las posibilidades que tiene la digitalización informática, pensamos que se ha obtenido un trabajo bastante agradable y con una relativa buena calidad.

¿Cuales fueron las fases de producción de Amazonia ?

Bueno, por supuesto hay una primera fase que es la de proyecto, quizá la más larga, pero lo primero es convencerse así mismo de que, meterse en una aventura de este calibre tiene futuro, y a partir de ahí hay que preparar muy bien qué materiales se van a llevar para grabar, porque aunque la selva es un medio hostil para el hombre lo es mucho más para la electrónica, y por ello hay que preparar muy bien los equipos; tienen que ser ligeros para andar por aquellas zonas.

Entonces Joaquín y José Antonio compusieron el equipo técnico, si bien José Antonio fue un mes antes para prepararlo.

A todas estas zonas hay que ir con un guía que las conozca. Amazonia dicen que es el país de los imprevistos: siempre salen las cosas de otra manera a como las tenías pensadas, pero bueno también es un aliciente en ese tipo de viajes. El guión que teníamos preparado era más adecuado para los audiovisuales, pero como había tanta cantidad de material, ¡30 horas de Betacam !, y bastante información recogida por escrito, fue muy difícil adaptarlo al CD-ROM.

A la hora de realizarlo lo principal fue crear un programa nuevo, es decir, diseñar el programa, como se iba a comportar, que interfaz debía tener, a donde debía llevar una pantalla tras otra, ese tipo de programación fue quizás lo más complicado, una vez que ya estaban tomadas las imágenes. Cuando todo estuvo decidido, Javier lo materializó, creemos que con bastante éxito, y ahí empezó el trabajo de rellenarlo: digitalizar esas imágenes en Betacam, grabar con locutores profesionales, sonorizar las locuciones con música original compuesta allí en el Amazonas, escribir los textos de los documentales, y por supuesto las informaciones que salen a pie de cada imagen. Un trabajo bastante complejo que hicimos en un tiempo bastante corto porque fue muy intenso. Había

días, más bien noches, que nos daban las siete de la mañana y aquí estabamos digitalizando sonido, todo con la idea de que pudiera salir al mercado cuanto antes.



¿La idea original es vuestra o fue un encargo ?

Es producción propia de Charmed, total y absolutamente, desde la idea original hasta el diseño de las cajas. No hemos estampado los discos porque eso es tarea de la fábrica, pero todo desde el principio hasta el final lo hemos realizado en esta empresa.

Es voluntad de la empresa invertir parte de los beneficios en algo de producción propia. El trabajar para otros clientes y hacer producciones para los demás tiene su aliciente, y es para lo que ha nacido Charmed, pero creíamos que nuestro interés o nuestra capacidad de hacer cosas, podía llevarnos a hacer algo muy diferente sin las presiones o condicionantes que habitualmente tiene cualquier producción para otra empresa, lógicamente.

¿Qué programas además de Amazonia habéis producido?

Bueno, el primero fue Metrópolis, una guía interactiva de calles, en principio para Madrid, y posteriormente incluso se ha hecho una versión para León. Es un programa que en principio nosotros no comercializamos pero que nos es muy útil para enseñar, captar nuevas ideas y nuevos clientes.,

Hay un proyecto de fusionar Metrópolis con el programa que ha servido de base a Amazonia para crear unos Puntos de Información, unas bases de datos y de imágenes bastante completas y muy útiles como información turística, por ejemplo.

¿Qué software habéis empleado?

En un principio Metrópolis, que fue lo primero, se hizo con Borland Pascal, que es un lenguaje orientado a objetos, con posibilidad de generación de código Windows. Luego Amazonia y "Sopa de Letras", que es otro programa que hemos producido, los implementamos en Visual Basic. Estas son las herramientas de programación que hemos utilizado.

¿El diseño del interfaz lo habéis pensado con algún fin didáctico? ¿Cómo fue el diseño?

El interfaz es lo más importante y lo que se ha cuidado más dentro de la programación, para que sea fácil y rápido. Luego además hemos querido incorporar la idea de meter cosas nuevas que no hay en otros programas, o al menos no las hemos visto, como puede ser el interfaz a pantalla completa siempre, esto es, que los controles se redimensionen en tiempo real, para que en función de la resolución de la tarjeta de vídeo se tenga siempre la máxima calidad, que es algo que normalmente no se ve en los programas multimedia.



¿Cuánto tiempo le dedicasteis a la programación desde que tuvisteis toda la información hasta que lo compusisteis todo?

Fue cosa de tres meses, más o menos, a un ritmo muy grande, pero también es verdad que Amazonia puede servir de base para otros programas. En un principio no disponíamos de ninguno de los recursos y la generación de transiciones de pantalla, de barras de estado, todo lo demás ha habido que hacerlo nuevo. Por ejemplo el retoque de imágenes y texto llevó un mes.

¿El desarrollo en Visual Basic ha facilitado la labor de programación respecto a otros lenguajes más tradicionales?

¡Muchísimo más! Metrópolis no se hizo en Visual Basic, un poco por desconocimiento y por que BASIC sonaba a principiante, por lo que se empleó Pascal orientado a objetos. Si se hubiera hecho con Visual Basic se habría tardado una porción del tiempo que se empleo con Borland Pascal.

¿La incorporación de los lenguajes de diseño visual es un gran avance para el desarrollo de las aplicaciones multimedia ?

Claro, porque además de facilitarte mucho la programación hay muchos elementos gráficos ya creados y bueno, el incorporar elementos multimedia tipo vídeo y sonido, es muy fácil, te lo dan prácticamente hecho, y es jugar a un nivel mucho más alto.

¿Qué proyectos estáis realizando actualmente?

Estamos preparando un CD-ROM sobre arte español, en realidad no es un proyecto nuevo, se trata una adaptación de un antiguo trabajo que se está adaptando a Windows, mejorando el interfaz gráfico y la calidad de la imagen en general.

Estamos en conversaciones con algunas empresas que están muy interesadas en producir materiales bastante cercanos o en la misma línea de lo que es ahora mismo Amazonia, o lo que pudo haber sido Metrópolis, o incluso como decía José Antonio el tema de los Puntos de Información Interactivos.

Respecto a esto último aún está sin definir, e incluso, sin terminar de hacer algunos contactos. Se trata de clientes españoles y algunos extranjeros. Es posible hacer algún trabajo para el gobierno de Amazonia, en concreto para la ciudad de Manaos, sobre un punto de información turístico bastante complejo. ■

Miguel Angel Alcalde

INSTRUCCIONES ARITMÉTICAS

Emilio Postigo

Si el ordenador nació con alguna finalidad, ésta fue el cálculo aritmético. En este capítulo vamos a estudiar las instrucciones, que para este fin tiene el procesador, y aquellas que están implicadas en su uso. Veremos que a partir de estas instrucciones "elementales" de suma, resta, multiplicación y división, es posible codificar subrutinas que efectúen cálculos en modo alguno triviales.

Es cierto que a partir del 486DX, el procesador incluye como parte de su repertorio de instrucciones, las que antes eran exclusivas de los coprocesadores aritméticos. Sin embargo es un hecho que no todos los ordenadores existentes hoy en día disponen de este elemento. Por este motivo es necesario, antes de estudiar el coprocesador, conocer el manejo y posibilidades de las instrucciones aritméticas convencionales. Así se estará en condiciones de hacer buen uso del lenguaje ensamblador.

INSTRUCCIONES ARITMÉTICAS

Son las listadas en la figura 1, en la que además se pueden ver los *flags* que son actualizados por ellas.

La primera que se va a tratar es la modesta instrucción *INC destino*, que se limita a sumar 1 a este operando (registro o memoria de uno o dos bytes de longitud). Su característica principal es que no actualiza el indicador de acarreo CF, ya que el operando se toma como entero sin signo. Se usa sobre todo dentro de bucles para incrementar registros base o índice, que se están empleando para acceder a posiciones de memoria. Su código de operación consta de un único byte cuando se incrementan registros de dos bytes. Por este motivo es más eficaz, por ejemplo, aumentar el registro SI con *INC SI* en vez de con *ADD SI, 0001*, que ocupa 3 bytes.

La instrucción *DEC destino* es empleada para restar 1 a un operando, y reúne las mismas características que la anterior.

SUMAR

La instrucción *ADD destino, fuente* realiza la operación destino=destino+fuente. Su empleo es trivial cuando se suman números sin signo de 16 bits, cuyo resultado está acotado por el valor 65535 (lo que en lenguaje C se denomina *unsigned int*), o números sin signo de 8 bits con un resultado máximo de 255 (llamados en C *unsigned short int*). Sin embargo, dado que esta instrucción está limitada en cuanto al tamaño de los operandos que puede utilizar (16 ó 32 bits, dependien-

FIGURA 1

Instrucciones aritméticas

Instrucción	OF	SF	ZF	AF	PF	CF
ADD destino, fuente	x	x	x	x	x	x
ADC destino, fuente	x	x	x	x	x	x
INC destino	x	x	x	x	x	-
SUB destino, fuente	x	x	x	x	x	x
SBB destino, fuente	x	x	x	x	x	x
DEC destino	x	x	x	x	x	-
MUL factor	x	?	?	?	?	x
IMUL factor	x	?	?	?	?	x
DIV divisor	?	?	?	?	?	?
IDIV divisor	?	?	?	?	?	?
NEG destino	x	x	x	x	x	x

(-) No afectada (x) Actualizada según definición
(?) Indeterminada

Cuando se estudia el lenguaje ensamblador es muy habitual comenzar explorando las enormes posibilidades que ofrece a la hora de manejar los recursos del sistema. Sin embargo, algunos elementos esenciales de la programación, como el cálculo aritmético, suelen ser descuidados.

do del modo de trabajo escogido), se plantea el problema de cómo operar con datos de mayor longitud que la establecida por la propia instrucción.

Supongamos que se desean sumar dos números sin signo de 32 bits de longitud. El procedimiento para efectuar la suma consistirá en dividirlos en dos partes (baja o *low* y alta o *high*) de 16 bits cada una. Se comenzará sumando las dos partes bajas y almacenado el resultado en el operando correspondiente. Ahora, para efectuar la suma de las partes más significativas, se deberá tener en cuenta la posibilidad de que se haya producido un acarreo (que nos hayamos "llevado 1") en la suma de los dos fragmentos menos significativos. Esto se sabe a través del *flag* de acarreo (*flag* CF). Por lo tanto será necesario efectuar la suma pendiente teniendo en cuenta el valor de este *flag*, y esto es lo que hace la instrucción ADC. Esta instrucción suma los operandos destino, fuente y el *flag* CF, dejando el resultado en el operando destino. Por este motivo, para efectuar correctamente nuestra suma de 32 bits, las partes altas han de sumarse con la instrucción *ADC*. Es obvio que, entre las dos sumas, no se puede alterar de forma arbitraria el contenido de este *flag*, so pena de producir un resultado incorrecto. En el listado 1 se muestra un ejemplo de estas acciones. Obsérvese que entre la primera suma y la segunda, sólo se ha ejecutado la instrucción *MOV*, que no modifica el valor de ningún *flag*.

Dado que se trabaja con operandos de 32 bits, se ha supuesto que el resultado tendrá este tamaño. De no ser así habría que tener en cuenta una posterior suma con *ADC*, aunque, de darse esta posibilidad, lo más cómodo es trabajar con un tamaño de datos mayor.

¿Qué ocurre si se están sumando números con signo? El principal problema que se da con este tipo de datos, como ya sabemos, es el del desbordamiento (*overflow*). Es decir, que al efectuar una operación de suma entre dos números, se rebase la capacidad de representación del sistema, hecho que puede ser advertido mediante el *flag* OF.

Cuando se trabaja con números sin signo y datos de -*n*- bits, el intervalo de números representables va de 0 a 2n-1. Si se utiliza la lógica de signo el intervalo abarca desde el valor -2n-1 hasta 2n-1-1. Con operandos de 8 bits, por lo tanto, se pueden representar números desde -128 a 127; trabajando con 16 bits, desde -32768 a 32767. Si, por ejemplo, se efectúa el cálculo 7FFFh + 0002h (32767+2), el *flag* OF tomará el valor 1, indicando que el resultado de la operación (32769) no puede ser representado empleando la lógica de signo.

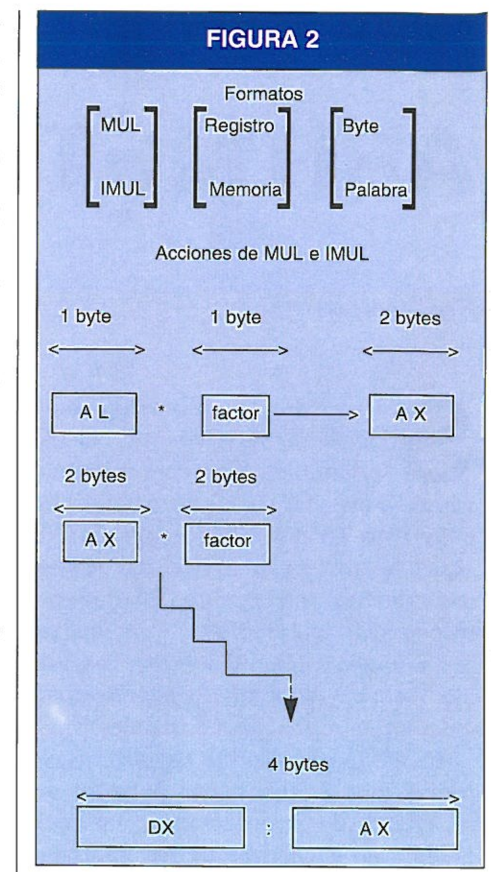
LISTADO 1

```
...
.DATA
  ...
  X DWORD ?
  Y DWORD ?
  Z DWORD ?
  ...
.CODE
  ...
 ; Se efectúa la operación Z = Y + X

  mov ax,   X[0]    ; Se comienza por los fragmentos de 16 bits
  add ax,   Y[0]    ; menos significativos
  mov Z[0], ax

  mov ax,   X[2]    ; Y se finaliza sumando la parte alta de los
  adc ax,   Y[2]    ; datos teniendo en cuenta el posible acarreo
  mov Z[2], ax      ; de la suma anterior
  ...
```

FIGURA 2



En efecto: para la C.P.U. se ha efectuado una suma de números positivos y el resultado es negativo. La solución estriba en emplear datos de mayor longitud, que permitan disponer de un mayor rango de representación.

Cuando se trabaja con números sin signo, el hecho de haber rebasado la capacidad del operando viene dado por un 1 en CF. Este *flag*, sin embargo, se puede ignorar cuando se trabaja con números con signo. En el caso de la suma FFFFh + FFFEh (-1 + (-2)) el resultado es FFFDh (-3) con CF=1. Ahora bien, el resultado es de todo punto correcto para números con signo y el valor de este *flag* se ignora.

Otra situación de interés es la de sumar operandos de distinta longitud. Trabajando con números sin signo se puede suponer que la parte alta del dato más corto es cero. Con operandos con signo la cosa cambia: si el número es negativo la parte alta ha de rellenarse con unos, y con ceros en caso contrario. Para solventar este problema de forma cómoda se dispone de las instrucciones *CBW, CWD. CBW* convierte un número de un byte almacenado en AL, en una palabra almacenada en AX,



LISTADO 2

```
...
DATA
...
   X SBYTE   ?
   Y SWORD   ?
   Z SDWORD  ?
      ...
CODE
...
; Se efectúa el cálculo Z = Z - Y + X

  MOV AL,  X   ; El valor con signo de X se expande primero a 2 bytes
  CBW          ; y, después a 4, quedando en los registros DX:AX
  CWD

  ADD Z[0], AX   ; Se suma la variable X sobre la variable Z
  ADC Z[2], DX

  MOV AX,  Y   ; Se transforma la variable de 2 bytes Y en otra de 4,
  CWD          ; con el mismo valor y almacenada en DX:AX

  SUB Z[0], AX   ; Se resta Y de Z, finalizando el cálculo
  SBB Z[2], DX
  ...
```

teniendo en cuenta el bit de signo. De esta forma, si se tiene en AL el valor 0Fh (=15), la instrucción *CBW* dejará en AX el número 000F (=15). Si se almacena en AL el número 80h (=-128), la ejecución de esta instrucción dejará en AX el valor FF80h (=-128). De la misma manera, *CWD* transforma una palabra almacenada en AX en una doble palabra en los registros DX:AX, expandiendo el bit de signo del dato guardado en AX en el registro DX. En el listado 2 se muestra cómo emplear estas instrucciones a la hora de efectuar el cálculo Z=Z-Y+X.

RESTAR

Para restar números se emplean las instrucciones *SUB y SBB* de manera similar a *ADC* y *ADD*. *SUB* se empleará para realizar la primera parte de la resta entre dos números, la correspondiente a las partes menos significativas. *SBB* se usará en el caso de que los datos posean un tamaño mayor que el máximo abarcable por la instrucción *SUB*. Ésta efectúa el cálculo destino=destino-fuente, mientras que *SBB* tiene en cuenta el valor del *flag* CF y hace la operación destino=destino-(fuente+CF). Las consideraciones realizadas para las operaciones de suma acerca de la longitud y el signo de los operandos, se pueden extender a estas dos instrucciones.

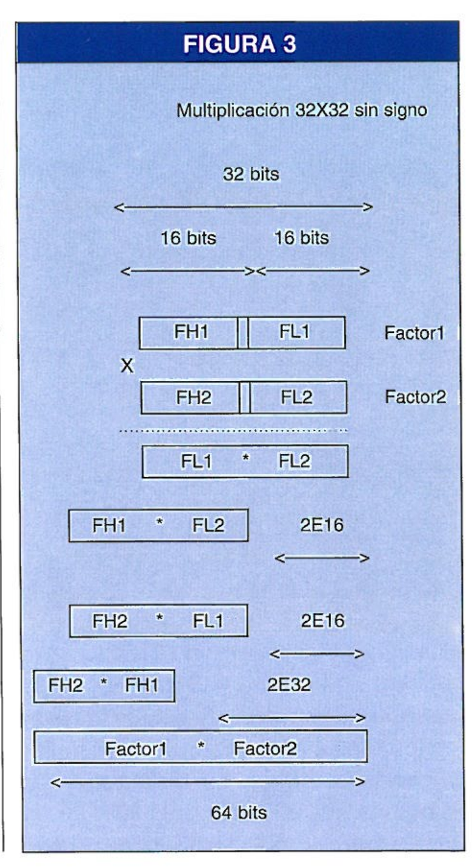
MULTIPLICAR

Para multiplicar, el procesador dispone de las instrucciones *MUL* e *IMUL*. En la figura 2 se muestra el formato de estas instrucciones y las operaciones que realizan automáticamente. Como se puede apreciar, sólo es necesario especificar un factor, ya que el otro está predeterminado en función del tamaño del operando fuente. Si éste es de un byte, el otro factor es el registro AL, y el resultado de la operación queda en el registro AX. Si el operando fuente es de dos bytes, el otro factor es el registro AX, y el resultado de la operación se almacena en los registros DX:AX. La parte más significativa en el primer registro y la menos significativa en el segundo.

La diferencia entre estas dos instrucciones estriba en que *MUL* trabaja con operandos sin signo, mientras que la C.P.U. interpreta éstos como con signo cuando se emplea la instrucción *IMUL*. Para ver la diferencia consideremos el siguiente caso: en los registros AL y BL se deposita el valor FFh y, a continuación, se ejecuta la instrucción *MUL BL*. El procesador realiza la operación 255X255=65025, y almacena en AX el número FE01h. Si en lugar de *MUL* se emplea *IMUL*, la cantidad FFh se interpreta como -1, y la operación efectuada es (-1)X(-1)=1, quedando en el registro AX el valor 0001h, como se puede comprobar inmediatamente desde el *debug*.

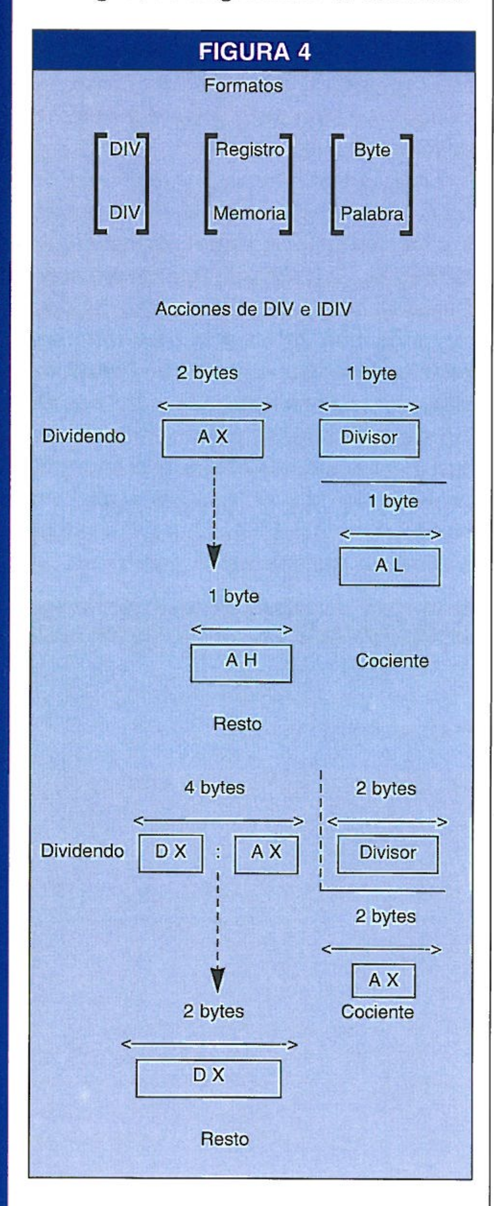
Estas instrucciones actualizan los *flags* CF y OF. Cuando después de una instrucción *MUL* la parte alta del resultado es 0, CF y OF son 0. En caso contrario ambos indicadores valen 1. Con la instrucción *IMUL*, si la parte alta del resultado es sólo la extensión del bit de signo de la parte baja, CF y OF son 0. En caso contrario ambos indicadores son 1. Si acudimos al ejemplo anterior, en el caso de la multiplicación sin signo, al ser AH=FEh, CF y OF valdrán 1. Para la multiplicación con signo, al

FIGURA 3



tener AH=00h, CF y OF valdrán 0. De esta forma el programa en curso puede "saber" si la parte alta del resultado tiene o no interés, de manera que el código pueda obviar algunas situaciones y ser más eficaz.

Como se ha visto, con estas instrucciones se pueden multiplicar entre sí números de 16 bits como máximo (no estamos trabajando con el modo de 32 bits). Si los operandos que es necesario multiplicar son, por ejemplo, de 32 bits sin signo, se fragmentan en dos mitades de 16 bits de manera idéntica a lo ya visto en la suma y la resta. A continuación se emplea el método basado en la propiedad distributiva de la multiplicación mostrado en la figura 3. La subrutina MUL32X32, incluida en el programa EJEMPLO1.ASM, constituye una posible implementación de este método.



Si los números de 32 bits poseen signo, el método de la subrutina anterior no funcionaría. En efecto: el bit de signo es el más significativo de toda la cadena de bits que constituye un operando. Si éste está fragmentado, el procesador no puede abarcarlo por entero en una sola instrucción, y la instrucción *IMUL* fracasa al efectuar productos parciales. La solución más cómoda consiste en obtener el complemento a 2 de aquellos operandos que sean negativos. Así se obtienen dos factores sin signo que pueden ser multiplicados por la subrutina anterior. Cuando ésta finaliza su trabajo, el resultado se mantiene como está en el caso de que los dos números originales fueran ambos positivos o negativos (resultado positivo, por lo tanto), o se complementa si sus signos eran opuestos (resultado negativo). Para obtener el complemento a 2 de un número se pueden emplear las instrucciones *NEG* o *NOT*, cuyo funcionamiento se explica dentro de EJEMPLO1.ASM, donde está incluida la subrutina IMUL32X32. Esta se apoya en MUL32X32 para realizar parte de su tarea.

DIVIDIR

Esta acción se puede realizar mediante las intrucciones ***-DIV y DIV-.*** La primera divide operandos con signo y la segunda operandos sin signo. En la figura 4 se pueden ver los formatos de estas instrucciones, y un resumen de las acciones que llevan a cabo. De la misma forma que en la multiplicación, la instrucción sólo va acompañada por un operando. Este operando es el divisor y mediante su longitud determina cuál es el operando dividendo. Si aquélla es de un byte el dividendo es AX. Si es de dos bytes, el dividendo está constituido por DX:AX.

Una vez efectuada la división, el cociente queda en AL y el resto en AH si el divisor era un operando de un byte. En caso contrario el cociente se almacena en AX, y el resto en DX.

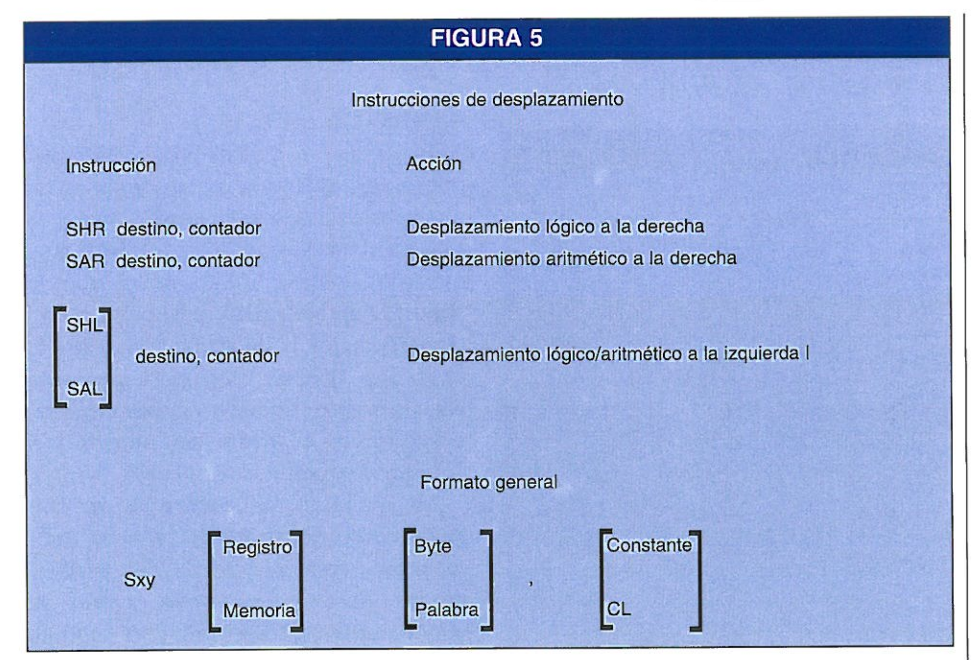
Ya vimos en el artículo anterior que esta instrucción podía dar lugar a la ejecución de una excepción, llamada habitualmente interrupción de división por 0 cuyo vector de interrupción es el 00h. Esto no significa que sólo cuando el divisor sea cero se vaya a ejecutar esta excepción. Esta se producirá siempre que el cociente de la división exceda el tamaño del operando destinado a tal fin. Si en AX se tiene, por ejemplo, el valor 100h y en BL el valor 01h, el cociente de la división *DIV BL* no cabrá en el operando AL y se generará la correspondiente llamada a *INT 0*. Si un programa va a emplear la instrucción de dividir con datos no controlados por el programador, éste no puede exponerse a que en cualquier momento se produzca un problema de este tipo y el programa se interrumpa. Por este motivo debe codificar un tratamiento apropiado del desbordamiento por división que sustituya al del sistema, haciendo que el vector 00h apunte a él, y manteniéndolo activo durante la ejecución del programa.

El tratamiento, como se puede ver en EJEMPLO2.ASM, consiste básicamente en fragmentar el dividendo, y efectuar divisiones parciales comenzando por los dígitos más significativos. Los restos que se obtienen se van acumulando sobre el fragmento de dividendo, con el que todavía no se ha trabajado. Dado que el divisor puede ser cualquier operando de 1 ó 2 bytes, el procedimiento ejemplo supone, para ganar claridad, que la instrucción que puede producir problemas es *DIV BL*.

INSTRUCCIONES DE DESPLAZAMIENTO

Por programas ejemplo de otros artículos hemos conocido estas instrucciones, cuyo formato se muestra en la figura 5. Su funcionamiento general consiste en desplazar, hacia la derecha (*SAR* y *SHR*) o hacia la izquierda (*SHL*) y a través de CF, los bits del operando destino el número de veces indicado por el contador, que puede ser el registro CL o bien una constante. Las posiciones que quedan libres son rellenadas con ceros en el caso de *SHR* y *SHL*, y con el valor original del bit de signo en el caso de *SAR*.

Al margen de las posibilidades que ofrecen de manipular bit a bit una variable, estas instrucciones pueden ser empleadas para efectuar multiplicaciones y divisiones sencillas de un



número por potencias enteras de 2. De la misma forma que añadir *n* ceros a un número decimal equivale a multiplicarlo por 10 * n, añadírselos a un número binario es lo mismo que multiplicarlo por 2 * n. Si en el registro AL se tiene el valor 04h, al ejecutar la instrucción *SHL AL, 1* se desplaza el contenido de este registro un lugar a la izquierda. El dígito más significativo pasa a el *flag* CF, que pierde su antiguo valor, y se añade un cero por la derecha, llegando AL a valer 08h. Si el valor de AL hubiera sido por ejemplo FFh (-1), al ejecutar la instrucción anterior habríamos obtenido como resultado FEh (-2). En ambos casos se ha duplicado el contenido de AL, por lo que se puede comprobar que, con cierto cuidado, se puede emplear *SHL* para multiplicar números, con signo o sin él, por potencias de 2.

Análogamente, *SAR* y *SHR* se emplean para dividir números con y sin signo, respectivamente, entre potencias de dos. Para verlo supóngase que en AL se encuentra almacenado el número FEh. Al ejecutarse *SHR AL, 1*, se pasa a tener en este registro el valor 7Fh, que es la mitad del original tomado como número sin signo. En caso de haberse ejecutado *SAR AL, 1* se habría llegado al resultado FFh, es decir, se habría pasado de -2 a -1. La figura 6 muestra gráficamente los pasos implicados en la ejecución de estas instrucciones.

El uso aritmético de las instrucciones que se acaban de comentar estuvo motivado en su día por la extraordinaria lentitud de las instrucciones *MUL*, *DIV*, *IMUL* e *IDIV* al ejecutarse en el 8086 (requerían entre 70 y 160 ciclos de reloj, según los operandos y la instrucción que fuera). Esto obligaba a emplearlas lo menos posible, y en cálculos que no pudieran ser optimizados mediante trucos basados en otras instrucciones. La aparición del 80286, redujo considerablemente el tiempo de ejecución de las instrucciones de multiplicación y división, pero no tanto como para "vencer" a las instrucciones de desplazamiento a la hora de efectuar multiplicaciones y divisiones por potencias de 2.

Dado que en muchos programas estas operaciones son habituales, es muy conveniente conocer y entender su manejo; esto evitará tener que emplear *MUL* o *DIV* en situaciones en las que su uso equivale a "matar moscas a cañonazos".

ARITMÉTICA DECIMAL

Después de ver las instrucciones aritméticas y sus peculiaridades, vamos a conocer la manera de manejar números decimales. Si se echa un vistazo al repertorio de instrucciones del procesador, se comprueba que existe un grupo de ellas, expuesto en la figura 7, dedicado al ajuste decimal después de operaciones de suma, resta, multiplicación y división. Esto no significa, sin embargo, que la C.P.U. sea capaz de "pensar" en decimal a la hora de efectuar ciertos cálculos. Para la máquina cualquier cantidad almacenada dentro de ella es binario puro, y es tarea del programador asignarles signo o formatos específicos.

Lo que estas instrucciones de ajuste permiten, es convertir el resultado de una operación a decimal empaquetado o desempaquetado, después de una operación muy simple efectuada con números almacenados en estos formatos.

Se dice que una cantidad se encuentra almacenada en formato decimal desempaquetado (incluso en ASCII, abusando de la terminología) cuando cada byte de esa zona de la memoria almacena un dígito decimal. En algunos textos se recalca el hecho de que los 4 bits menos significativos constituyen el dígito en sí, mientras que los 4 de mayor peso almacenan el signo del número o bien una cantidad característica llamada "*zona*", que suele tomar el valor Fh. Así, es habitual decir que un byte que guarde el valor F3h almacena el dígito 3 en formato decimal desempaquetado. Sin embargo, para armonizar con las instrucciones de ajuste del procesador se empleará el convenio de que la zona vale siempre 0h, y que la asignación de signo corre por cuenta del programador.

Una vez aclarado este punto, comencemos con la instrucción *AAA* o *ajuste ASCII tras la suma*. Ésta se encarga de convertir en decimal desempaquetado el resultado de sumar un dígito decimal almacenado en AL, con cualquier operando de un byte. El resultado se deja en el registro AX. Como ejemplo, supóngase la secuencia de instrucciones *MOV AL, 08h / ADD AL, 05h / AAA*. Después de la instrucción de suma el valor almacenado en AL es 0Dh, ya que el procesador los ha sumado como datos binarios que son. El efecto de la instrucción *AAA* consiste en cambiar de base esa cantidad, dejando en AL el dígito 3, y en AH el 1. Evidentemente, si se desea sumar números de más de un dígito de longitud, será necesario "arrastrar" el valor que queda en AH, y tenerlo en cuenta en la suma de dígitos de mayor peso.

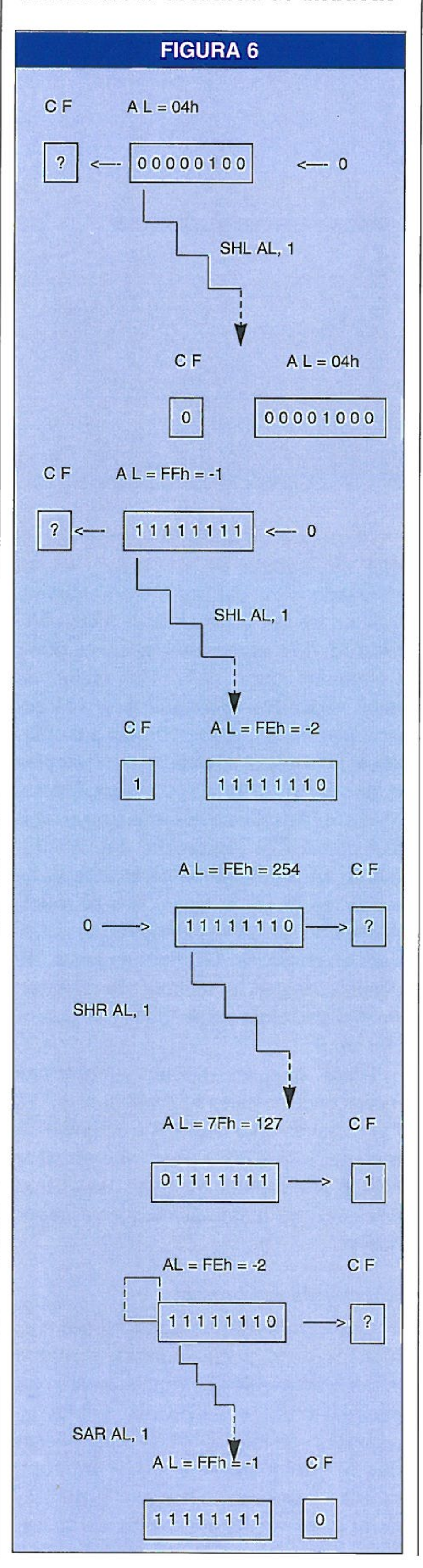
La instrucción *AAS* es el ajuste ASCII tras la resta, y trabaja de una forma muy

similar. Si se ejecutan las instrucciones *MOV AX, 0108h / SUB AL, 09h* (18 - 9 en decimal desempaquetado), el valor que queda en AX será 01FFh. Al introducir el ajuste *AAS*, este valor pasa a ser 0009h, que es el esperado al restar en base decimal.

Con *AAM* o *ajuste ASCII tras la multiplicación,* es posible multiplicar dos dígitos decimales, uno de ellos en AL, dejando el resultado en AX. A modo de ejemplo supongamos que se desea multiplicar 8 x 4. La secuencia de instrucciones oportuna será: *MOV AL, 08h / MOV BL, 04h / MUL BL / AAM.* El efecto de esta última instrucción es muy similar al de *AAA*: El valor hexadecimal almacenado en AL es convertido a decimal, empleándose AL para almacenar el dígito menos significativo del resultado y AH el de mayor peso. Por último, cuando se desea dividir un número decimal entre otro se puede acudir a *AAD* (ajuste ASCII anterior a la división), para obtener un cociente decimal. Esta instrucción ha de ser ejecutada antes de la división. Su efecto no es otro que el de convertir en binario un número de dos dígitos decimales almacenado en AX, de forma que después se efectúe una división normal. El cociente de esta división se guarda en AL y se convierte en decimal desempaquetado con la ya vista *AAM.* La secuencia de instrucciones *MOV AX, 0301h / MOV BL, 02h / AAD / DIV BL / AAM* puede ser utilizada para comprender el funcionamiento de una división decimal. Obsérvese que el resto, valor de AH después de *DIV BL*, no es respetado al efectuar la última conversión, por lo que será necesario copiarlo en otra variable si se desea utilizar posteriormente.

Las otras dos instrucciones de ajuste son *DAA* o *ajuste decimal para la suma,* y *DAS* o *ajuste decimal para la resta.* Estas instrucciones "arreglan" el resultado de una operación de suma o resta cuyo resultado está almacenado en AL. Para que el ajuste tenga sentido, el cálculo debe ser realizado con números en formato decimal empaquetado.

Almacenar un número de esta forma significa guardar en cada byte dos dígitos decimales. Por ejemplo, los valores 34h, 01h y 99h pueden ser interpretados, llegado el caso, como números en formato decimal empaquetado. Operar con ellos usando las instrucciones vistas arriba es muy sencillo. Como ejemplo, considérese la secuencia de instrucciones *MOV AL, 57 / ADD AL, 38 / DAA / SUB AL, 13 / DAS.*



CAMBIO DE BASE

El hecho de que el procesador disponga de recursos para operar con números decimales, es debido a que en este formato, los números adquieren su máxima precisión. Pensemos que la simple cantidad 0,3 carece de una representación finita en binario, y por lo tanto, un cálculo efectuado con este valor siempre produciría cierta imprecisión, mayor o menor en función del tamaño de operando empleado.

Sin embargo, la codificación de cálculos complejos a partir de esos rudimentarios recursos puede ser terriblemente lenta y engorrosa, como se puede suponer. En el próximo capítulo veremos que el coprocesador aritmético puede trabajar de forma rápida y eficiente a partir de números en formato decimal empaquetado.

De todas formas, las operaciones comunes, cuando no requieren un grado excesivo de precisión, pueden ser efectuadas con números binarios, en coma fija o flotante. Como los datos numéricos que necesita un programa suelen llegar a través del teclado, entran dentro de las variables en formato decimal desempaquetado con un valor de zona igual a 3h. Para poder operar con ellos en binario será necesario convertirlos a esta base, y eso es lo que se va a estudiar desde este momento.

DE DECIMAL A BINARIO

Para pasar un número de base diez a base hexadecimal debe aplicarse el conocido teorema fundamental de la numeración (TFN). Como ya se sabe, los dígitos correspondientes a la parte entera del número, se multiplican por potencias de exponente positivo de la base original, mientras que los pertenecientes a la parte fraccionaria se corresponden con exponentes negativos. Por ejemplo, el número 34,17 decimal se pasaría a hexadecimal mediante la expresión 3xA1+4xA0+1xA-1+7xA-2.

El cálculo correspondiente a las potencias positivas es inmediato: 22. Sin embargo, por lo que respecta a la parte fraccionaria se plantea el siguiente problema: ¿Cómo efectuar divisiones fraccionarias empleando las instruccio-

FIGURA 7

Instrucciones de ajuste decimal

Instrucción	Acción
AAA	Ajuste ASCII después de una suma
AAS	Ajuste ASCII después de una resta
AAM	Ajuste ASCII después de una multiplicación
AAD	Ajuste ASCII anterior a una división
DAA	Ajuste decimal para la suma
DAS	Ajuste decimal para la resta

nes del procesador, si éstas realizan divisiones enteras? Este problema se puede solventar de varias formas, pero la más breve es la siguiente.

En primer lugar se escoge una precisión máxima para la parte decimal, desechando dígitos si hay más de la cuenta o añadiendo ceros si faltan. Supongamos que, en nuestro ejemplo, se desea trabajar con 4 dígitos en la parte decimal: ésta será 0,1700.Ahora se expresa la parte decimal como un entero multiplicado por la potencia de diez apropiada: se llega a 1700x10-4.

Se aplica el TFN a la parte entera, y se expresan todos los números en hexadecimal: 6A4xA-4.

Llegados a este punto nos detenemos y observamos la última expresión. Realmente ¿Se ha adelantado algo con respecto al problema original? En éste se tenían dos divisiones no realizables y ahora sólo tenemos una, que tampoco se puede llevar a cabo, ya que el dividendo es menor que el divisor. El cociente de la operación va a ser cero con toda seguridad, y no se puede afirmar nada sobre la forma hexadecimal de la parte fraccionaria.

Sin embargo, ahora se puede reescribir la última expresión como 64A0000xA-4x10-4 (¡ojo! 10=dieciséis, ya que se opera en hexadecimal), y ahora ya se puede abordar la división. El cociente de la misma es la parte fraccionaria del número en hexadecimal hasta 10-4, y el resto es el error cometido con respecto al operando original debido al truncamiento del proceso. En este caso se tiene 0,2B85 y 0,00001E respectivamente. Ya se dijo antes que en un cambio de base las partes fraccionarias acumulaban errores. En este caso, el error será del orden de 10-4 (0,000015). Si se desea disminuirlo, bastará con multiplicar el resto por 104 y dividirlo de nuevo entre A4, ampliando la parte fraccionaria del número. El haber escogido una parte fraccionaria inicial de cuatro dígitos se ha hecho con vistas a los cálculos posteriores. Dado que se va a emplear la instrucción *DIV* para efectuar las divisiones pertinentes, se ha buscado la máxima potencia de diez menor que 65535, de forma que los cálculos sean rápidos, pero puedan llevarse a cabo dentro de operandos de 16 bits. Una vez obtenida la expresión binaria del número, se comprueba el signo de éste. Si es positivo, el proceso ha finalizado. Si es negativo se obtiene su complemento a 2 mediante las instrucciones *NEG* o *NOT*.En el archivo EJEMPLO3.ASM se encuentran las subrutinas DEC2HEXFRC, que se encarga de realizar los cálculos descritos mediante el ejemplo anterior, y DEC2HEXINT, cuya tarea es obtener la parte entera del número inicial en hexadecimal. Estas dos subrutinas se usan de forma combinada para transformar un número en formato decimal desempaquetado en un número binario con signo.

DE BINARIO A DECIMAL

Una vez pasados los números de decimal a binario, y efectuados los cálculos pertinentes, es necesario poder mostrarlos por pantalla en un formato comprensible. O sea, llega el momento de cambiar su representación interna de binario a decimal. Como en el caso opuesto, se dividirá el problema en dos partes: la obtención de la expresión decimal de la parte entera del número, y la de la parte fraccionaria. El primer objetivo se alcanza mediante el conocido método de las divisiones enteras sucesivas, que es el empleado por el procedimiento HEX2DECINT, definido dentro de EJEMPLO3.ASM. La parte fraccionaria, por otro lado, se obtiene multiplicando la parte fraccionaria original binaria por el número 0ah (diez), y tomando los dígitos que excedan el tamaño del operando. De este trabajo se encarga la subrutina HEX2DECFRC.

BIBLIOGRAFÍA

Microsoft Macro Assembler Programmer's Guide, V 6.0 y sucesivas, Microsoft Corporation

Microsoft Macro Assembler Reference, Microsoft Corporation

Scanlon, Leo J.: *80286, Programación ensamblador en entorno MS DOS.* EDICIONES ANAYA MULTIMEDIA, Madrid, 1988.

Quizá a alguna persona le haya sorprendido el empleo de estos métodos de cambio de base. Acostumbrada al empleo del TFN para pasar de cualquier base a base decimal, y al del método de las divisiones sucesivas para efectuar el paso contrario, habrá observado que las subrutinas ejemplo los usan exactamente a la inversa: el TFN para pasar de base decimal a hexadecimal y las divisiones enteras sucesivas para pasar de hexadecimal a decimal.

Esto no debe confundirnos. Lo que ocurre es que el TFN es empleado para pasar de una base "extraña" a una conocida, y el otro método para el proceso opuesto: pasar de una base conocida a una "extraña". El que una base sea conocida significa que se conocen las tablas de sumar, restar, multiplicar y dividir. Para los seres humanos la decimal suele ser la única base conocida y las demás extrañas, pero con el procesador ocurre de forma distinta: conoce muy bien el sistema de numeración hexadecimal, pero en el decimal se pierde. Este es el motivo de que las subrutinas de cambio de base trabajen en la forma que lo hacen.

PARA FINALIZAR

Como advertencia, es necesario señalar que el formato de los números empleado en este programa (coma fija con signo, partes entera y fraccionaria de 16 bits) tiene una finalidad exclusivamente didáctica, y no se corresponde con ninguno de los empleados por los lenguajes de alto nivel. Con él sólo se pretende construir un puente entre los números binarios enteros, y los números en formato de coma flotante, que serán estudiados en el siguiente capítulo, dedicado al coprocesador aritmético. ■

CONSTRUYENDO UN PROGRAMA

Miguel Angel Alcalde

En los dos capítulos anteriores de este curso se han visto los aspectos más generales que componen el "idioma" de un ordenador. La mejor forma de comprender realmente estos conceptos es dejar la teoría a un lado y pasar a la acción.

Antes de comenzar a escribir el programa que sirve como práctica en esta tercera entrega, es necesario crear desde el principio buenas costumbres para evitar caer en los errores y vicios que cometen quienes aprendieron a programar en tiempos en que esta profesión estaba considerada como "mágica".

¿QUÉ ES UN PROGRAMA?

Es una serie de ordenes, una forma de expresar acciones, un método para conseguir un resultado, y que lo realizará una máquina. Dicho de forma más poética, programar es digitalizar pensamientos.

Ahora bien, a la hora de decir cómo hay que hacer algo, es importantísimo tener en cuenta que ha de definirse de la forma más breve y concisa posible, ya que hay que aprovechar el magnífico recurso de los ordenadores: pueden repetir indefinidamente una acción y no se cansan.

Por lo tanto antes de lanzarse a codificar, hay que pensar qué se quiere hacer, qué se necesita para lograrlo y por último, cómo se construye.

Así pues se va a pasar directamente a crear este primer y sencillo programa teniendo en cuenta estas premisas y todas aquellas que surjan durante este proceso.

¿QUÉ PROGRAMA HACEMOS?

Como es imposible hallar uno que satisfaga a todos los lectores, se ha pensado que un juego sencillito sería lo más adecuado ya que es mucho más evidente y real que una base de datos o un conversor de ficheros.

El juego elegido es muy similar a uno "histórico", cuyo nombre seguro que es conocido por la mayoría de los lectores. Para evitar suspicacias se denominará Frontón y de esta forma se promociona también este deporte tan español como los toros.

El funcionamiento será muy sencillo. Se dispone de una raqueta que se mueve horizontalmente al pulsar dos teclas y de una pelota que rebota en los lados izquierdo y superior de la pantalla. Hay que intentar que la pelota no se salga de la pantalla golpeandola con la raqueta.

¿QUÉ SE NECESITA?

- Es imprescindible un compilador de Pascal ya que sin él no se puede hacer que funcione el programa.
- Un buen montoncito de papel y un lápiz para pensar y escribir las ideas que surjan.
- Si se puede, jugar un buen rato al Frontón. En su defecto una canica y un par de libros pueden simular el escenario donde tendrá lugar la acción.
- Como se está aprendiendo, un buen libro sobre el lenguaje de programación elegido, en este caso Pascal.
- Si es posible, conseguir un programa que haga lo mismo o parecido a lo que se intenta hacer para tomar ideas observando cómo funciona.

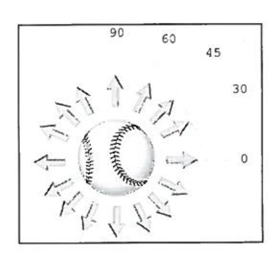
¿CÓMO SE CONSTRUYE?

- Se necesitará un compilador para traducir el programa escrito en el lenguaje de alto nivel empleado, Pascal, a código máquina, así como un editor de textos para escribirlo. Hace algunos años éstas eran las herramientas empleadas por todos los programadores, actualmente los entornos de desarrollo integrado (IDE) al facilitar el trabajo de los informáticos, son la configuración básica para crear programas.

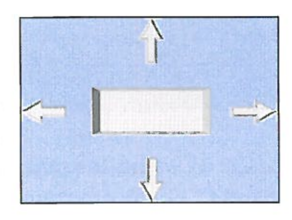
Las funciones que los IDE aportan son la edición, compilación y depuración del código sin tener que abando-

nar la herramienta, simplificando el manejo de estos recursos. Los compiladores "Turbo" de Borland son un buen ejemplo.

- Además es preciso documentarse para realizar un programa que cumpla con precisión todas las reglas y requisitos necesarios del objetivo. En este caso parece que todo está muy claro y las reglas son fáciles de implementar, pero si el ordenador no fuese capaz de calcular senos y cosenos para hallar la trayectoria tras el impacto, ¿cómo se solucionaría?. Antes de empezar a programar hay que prever cualquier cuestión que pueda surgir.



Las flechas indican la dirección que puede tomar la pelota tras rebotar en las paredes.



El cursor puede moverse en estas cuatro direcciones por toda la pantalla.

Está claro que el juego funcionará sobre modo texto, pero si se quisiera poner en modo gráfico 320 x 400, ¿cómo se activa este modo de vídeo?. Habría que conseguir un libro sobre la programación de la VGA o un ejemplar de Sólo Programadores para averiguarlo.

En este caso es muy importante saber como mover la "raqueta". Si se emplea el teclado puede ser necesario implementar la función de lectura de la tecla en ensamblador para que la respuesta sea más inmediata. También el ratón podría ser útil, y la verdad sea dicha, hay pocos libros que expliquen cómo se programa un ratón.

- Por último hay que pensar quién lo va a usar. El aspecto no puede ser igual si juega un niño o un adulto. En el primer caso habrá que poner muchos colores y adoptar un aspecto divertido, mientras que en el segundo una ambientación profesional será más adecuada.

COMENZANDO A PROGRAMAR

Quizás pueda parecer muy atrevido crear un juego sin haber hecho antes un programa de al menos tres líneas, pero lo que se intenta es hacer comprender que lo importante no es saber Pascal sino cómo se programa.

La programación, afortunadamente, tiene aún algo de artesanal por lo que es muy importante observar cómo transcurre el pensamiento de este programa para sacar impresiones propias y adaptarlas a cada mente, ya que para cada problema existen infinitas soluciones, tantas como personas piensen sobre él.

Por lo tanto interesa saber de nuevo que programa se iba a hacer: Un juego de Frontón.

- ¿Cuales van a ser las reglas?. Para que no ocupe toda la revista hay que elegir un conjunto muy pequeño que sintetice el espíritu de este juego:

1) El terreno de juego es toda la pantalla, y los bordes superior e izquierdo son la paredes.
2) La pelota debe golpear siempre en la pared superior. Podrá rebotar con los siguientes ángulos: 0, 30, 45, 60 o 90 grados. La velocidad de desplazamiento permanecerá constante.
3) El jugador podrá moverse horizontal o verticalmente.
4) La raqueta tendrá un tamaño de cuatro caracteres.

La fuerza imprimida a la pelota será constante y el efecto dependerá de la velocidad de desplazamiento lateral que posea.

5) El juego acaba cuando la pelota se sale de la pantalla por los lados derecho o inferior.

Seguramente hay algún aspecto que se ha quedado fuera o que sobra, pero en principio es una buena aproximación para el juego que se pretende hacer.

- Se han definido las reglas o normas de este juego. Una persona adulta quizás no necesitaría nada más para ponerse a jugar, pero un niño seguro que requerirá más datos para hacerlo, que le simplifiquen el aprendizaje, como por ejemplo una descripción de cómo se juega. Luego pasemos a explicar cómo se juega, a programar:

APROXIMACIÓN 1

Juego de Frontón 1:

Se tiene una raqueta y la pelota está en ella;
2) El usuario mueve la raqueta;
3) La pelota sigue su trayectoria;
Si se sale del campo entonces
acaba la partida;
Si ha golpeado la pared superior entonces
Si el usuario golpea la pelota con la raqueta
entonces la pelota sigue su trayectoria
en otro caso no la ha dado y la partida
termina;

Con estas instrucciones se aclara un poco pero siguen existiendo imprecisiones:

- En la línea 2 no queda muy claro cómo se mueve la raqueta.
- En la línea 3, la palabra trayectoria deja demasiados conceptos oscuros.

Para solucionar esto es necesario realizar una aproximación mayor, explicando más cada concepto:

APROXIMACIÓN 2

Juego de Frontón 2:

La raqueta se encuentra en la posición inicial;
La pelota está encima de la raqueta;
Si el jugador pulsa una tecla entonces
mueve la raqueta y la pelota adquiere trayectoria
dependiendo de cómo y qué tecla pulsó;
Mientras la pelota no se salga del terreno hacer
Calcular la posición de la pelota en cada
momento;
Si hay impacto con pared entonces
Calcular nueva trayectoria;
Si la pelota ha tocado la pared superior entonces
Si el jugador toca la pelota con la raqueta
entonces
Calcular nueva trayectoria dependiendo de cómo y qué tecla pulsó;
Fin mientras.

En principio este algoritmo parece correcto y aclara un poco más cómo suceden las cosas. Éste es un detalle importante, cuando se programa algo hay que definir cada regla o ley de ese pequeño mundo virtual, una tarea que hace sentir al programador que en realidad está construyendo un pequeño universo vivo.

Volviendo a estas hojas, es posible que algún avispado lector descubra un posible fallo en este código inicial, pero no importa, si existe seguro que aparecerá evidente en el momento oportuno.

De nuevo hay que profundizar el nivel de detalle, ya que aún es imposible codificar en Pascal:

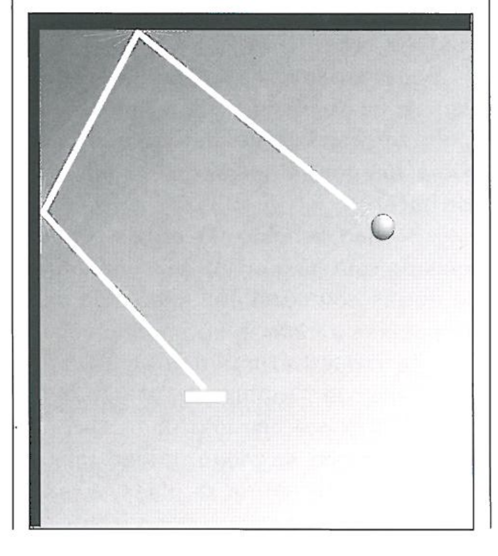
APROXIMACIÓN 3

Juego de Frontón 3:

Colocar la raqueta en la línea del fondo de la
pantalla;
Colocar la pelota en la línea superior a la raqueta,
justo en medio de ella;
Si el jugador pulsa un cursor entonces
1)Hacer que la trayectoria de la pelota sea
vertical;
Mover la pelota en la dirección adecuada;
Mover la raqueta en la dirección del cursor
pulsado.
Mientras la pelota no se salga del terreno hacer
Calcular la posición de la pelota en cada
momento;
Mover la pelota en la dirección adecuada;
2)Actualizar la posición de la raqueta leyendo
la tecla pulsada;
Si la pelota alcanza una pared entonces
Calcular nueva trayectoria dependiendo
del ángulo con el que tocó la pared;
Mover la pelota en la dirección adecuada;
Si la pelota ha tocado la pared superior entonces
Si el jugador toca la pelota con la raqueta
entonces
calcular nueva trayectoria dependiendo
de cómo y qué tecla pulsó;
Fin mientras.

De nuevo hay dos líneas que incorporan mayor definición:

En 1) se fuerza a que el saque sea siempre contra la pared superior o frontal, ya que así sucede en el juego real y porque facilita el comienzo del juego.



GLOSARIO DE TÉRMINOS

Algoritmo: Método o técnica para resolver un problema.
Código: Texto que representa un programa independientemente del lenguaje con que esté escrito.
Compilar: Traducir un código escrito en un lenguaje de programación a código máquina, directamente entendible por el ordenador.
Lenguaje: Conjunto de palabras y formas de expresión empleadas por el hombre para relacionarse con otros seres.
Programa: Formulaciones concretas de algoritmos abstractos basadas en representaciones y estructuras concretas de datos
Seudocódigo: Lenguaje de programación que se encuentra entre el lenguaje natural y el empleado para codificar en los ordenadores.
Traducir: Convertir un texto escrito en un idioma a otro.

BIBLIOGRAFÍA

Título: ALGORIMOS + ESTRUCTURAS DE DATOS = PROGRAMAS.
Autor: Niklaus Wirth.
Editorial: Ediciones del Castillo.

En 2) se ha introducido la "interactividad" en el programa al permitir al jugador moverse por la pantalla para golpear a la pelota.

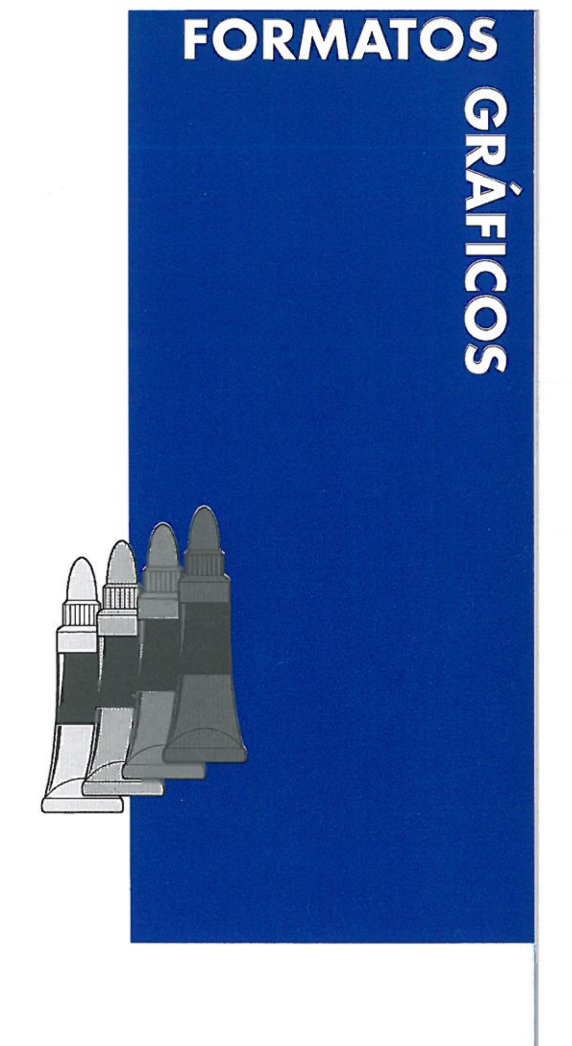
En principio una persona con ciertos conocimientos no necesitaría seguir profundizando en el diseño, ya que pasar de este seudocódigo a Pascal es bastante fácil si se saben resolver detalles como por ejemplo leer una tecla y calcular cuanto tiempo se ha mantenido pulsada, para conseguir el parámetro que imprimirá la dirección a la pelota cuando choque con la raqueta.

EN EL PRÓXIMO CAPÍTULO

En la siguiente entrega se comenzará a implementar el juego en Pascal. Mientras tanto se propone a los lectores que intenten continuar la definición del seudocódigo hasta que lleguen a operaciones básicas como sumar restar, leer o escribir. Quienes lo consigan pueden enviarlo a la Redacción de Sólo Programadores haciendo referencia en el sobre que va dirigido a la sección Curso de Programación Básica. ■

COMPRIMIENDO AL LÍMITE

Agustín Guillén



Continuamos la serie empezada unos números atrás sobre formatos gráficos, con el formato de moda del momento, uno que ofrece los imprescindibles 16 millones de colores (true color), unido a una compresión realmente asombrosa (valores superiores a 20:1 son habituales), se trata del formato JPEG. Sólo tiene una limitación, para obtener esos enormes valores de compresión, "modifica" sutilmente la imagen, descartándose su uso en las aplicaciones en las que se deba mantener una calidad bit a bit, o en las que no se pueda modificar ni un ápice la imagen original. El diseño de este formato está pensado para almacenar imágenes del "mundo real", también llamadas imágenes de tono continuo, como digitalizaciones o renderizaciones de alta calidad. Si se intenta almacenar pantallas del tipo de imágenes vectoriales o dibujos sencillos no realísticos, se observará como la compresión disminuye enormemente, y las modificaciones hechas sobre la imagen original por el algoritmo de compresión se observan a simple vista.



El formato JPEG está pensado para almacenar imágenes digitalizadas o renderizadas

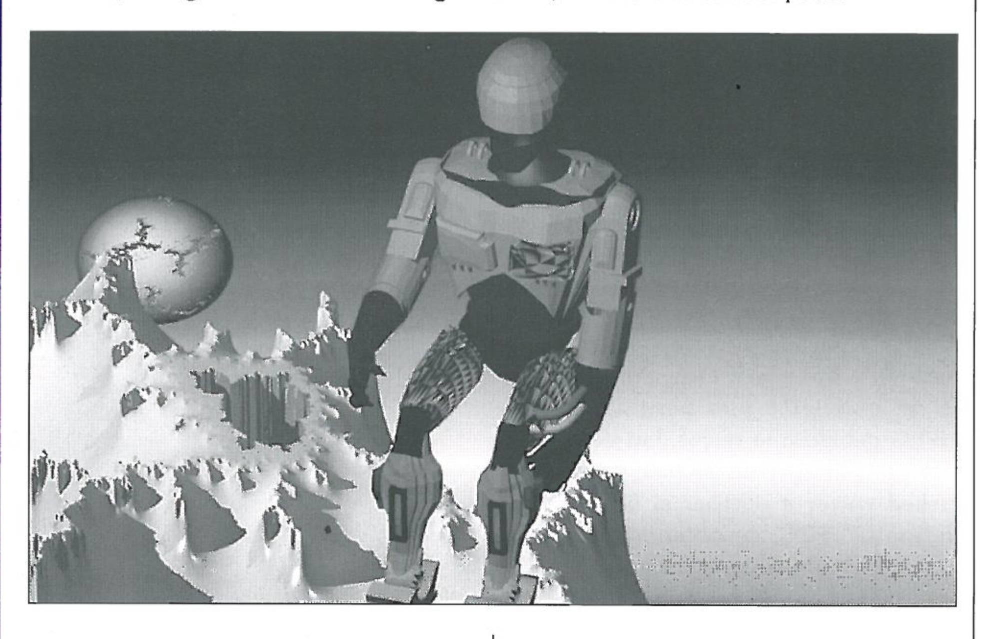
La abreviación JPEG viene de las iniciales de "Joint Photographic Experts Group". Se trata del grupo de expertos que definieron las bases de este formato. Y aunque son varios años los transcurridos desde su definición, y varias las versiones escritas, todavía su completa descripción se encuentra en fase de desarrollo.

Cuando se trabaja con formatos que modifican el modelo original, se debe tomar la precaución de guardar la imagen anterior, y así tenerla como base de partida para la posible conversión a otros formatos. Por ejemplo, si se convierte una pantalla GIF a JPEG debido a la necesidad de disponer de la imagen en 24 bits, y luego se desea volver a obtener el fichero GIF desde el JPEG, se obtendrá, además de la previsible disminución en la calidad de la imagen, un fichero de aproximadamente el doble de tamaño que el fichero original. Esto se debe a que el formato JPEG no funciona correctamente con imágenes que contengan zonas de un solo color y las modifica sutilmente con lo que el formato GIF, que sí que se especializa en zonas del mismo color, no puede volver a comprimirlas de nuevo. Existe un formato gráfico que aúna lo mejor de ambas filosofías de compresión, se trata del HSI JPEG, este formato detecta si existen zonas extensas del mismo color y "desactiva" la compresión JPEG. El problema es que este formato no es compatible, ni con el JPEG ni con el GIF. Otro formato muy utilizado, el estándar TIFF en su versión 6.0, incluye en su definición la compresión JPEG, pero debido a lo confuso y somero de su descripción, todavía no es muy utilizado por los desarrolladores.

¿Vale la pena sacrificar calidad para conseguir compresión? Como siempre, depende de la aplicación final de las imágenes; pero si nos fijamos en cualquier proceso para obtener o visualizar una imagen, está lleno de

La proliferación de tarjetas capturadoras de vídeo, de tarjetas VGA con 24 bits de color, y de impresoras capaces de mostrar 16 millones de colores, están condenando a muerte al rey de los formatos gráficos de intercambio: el formato GIF. Su fallo, sólo poder almacenar un máximo de 256 colores. Su sucesor, el formato JPEG no tiene ese límite, y ofrece además una compresión que asustaría a los mismísimos Lempel-Ziv & Welch (LZW).

pérdidas de calidad, y modificaciones del original: la obtención de la fotografía original está llena de mermas en la calidad, la digitalización de la imagen



mediante un scanner o similar tiene pérdidas, la visualización en el monitor tiene pérdidas y la impresión final ...

CODIFICACIÓN

El formato JPEG sólo puede almacenar imágenes de 24 bits (true color), utilizando tres canales para su almacenamiento o de escala de grises, usando sólo un canal.

La compresión JPEG consiste en una serie de complejas operaciones matemáticas, tales como: conversión del formato del color, transformación separada de coseno (DCT), cuantizaciones y codificación entrópica.

Los pasos necesarios para una compresión JPEG son los siguientes:

- Antes de empezar, se trabajará con datos puros (raw) del tipo RGB, o de escala de grises, por lo que si no están en este formato, habrá que prepararlos previamente.

- Conversión de los datos originales RGB en datos de color YCbCr (luminancia y dos crominancias), también conocidos como datos YUV. Este proceso se denomina color space conversion, y es realizado porque el ojo humano es mucho más sensible a la información sobre luminancia que a las crominancias, y si se separan es posible comprimir más estas últimas sin merma aparente de la calidad. Este paso se omitirá, si los datos de entrada son del tipo escala de grises. Si es necesario realizar un ajuste del nivel gamma, se realizará en este paso.

- Reducción de las muestras de entrada en componentes de color, este procedimiento es denominado submuestreo (subsampling). Un muestreo utilizado es: 2h:1v, 1h:1v, 1h:1v, e indica que el primer componente (luminancia) tiene el doble de muestras horizontales que los otros dos componentes (crominancias), y el mismo número de muestras verticalmente.

Los pasos siguientes se realizarán una vez para cada "barrido" del fichero de salida JPEG, o lo que es igual, una vez si el fichero es entrelazado o varias si es no-entrelazado.

- Creación de los MCU (ver glosario), o partición de la imagen en bloques de 8 por 8 pixels.

- Transformación DCT para cada MCU (bloque de 8 x 8). Este es el verdadero corazón del formato JPEG y es la que se encarga de la compresión propiamente dicha.

- Escalado cuantificado, y reordenado "zigzag" de los elementos de cada bloque de 8 x 8.

- Codificación aritmética o de Huffman de los bloques transformados.

- Creación de las cabeceras necesarias para un fichero JPEG.

Los necesarios para la descompresión de un fichero JPEG son éstos:

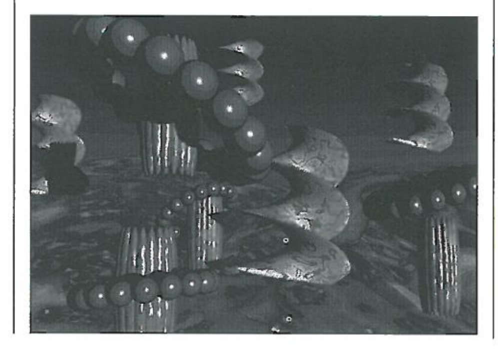
- Lectura del fichero JPEG, extrayendo los datos de la cabecera.

- Decodificación de la secuencia de datos, bien sea aritmética o del tipo Huffman.

- Escalado, cuantificado, y reordenado "zigzag" de los elementos de cada bloque MCU.

- Desensamblado de los bloque MCU (también reconstrucción de los bloques originales si el fichero fuera entrelazado).

Este formato modifica sutilmente la imagen, descartándose su uso en las aplicaciones en las que se deba mantener una calidad bit a bit



- Posible aplicación de un algoritmo de suavizado (smoothing) a cada bloque.

- Transformación inversa DCT para cada bloque.

- Reconstrucción de la imagen original, mezclando los componentes de color.

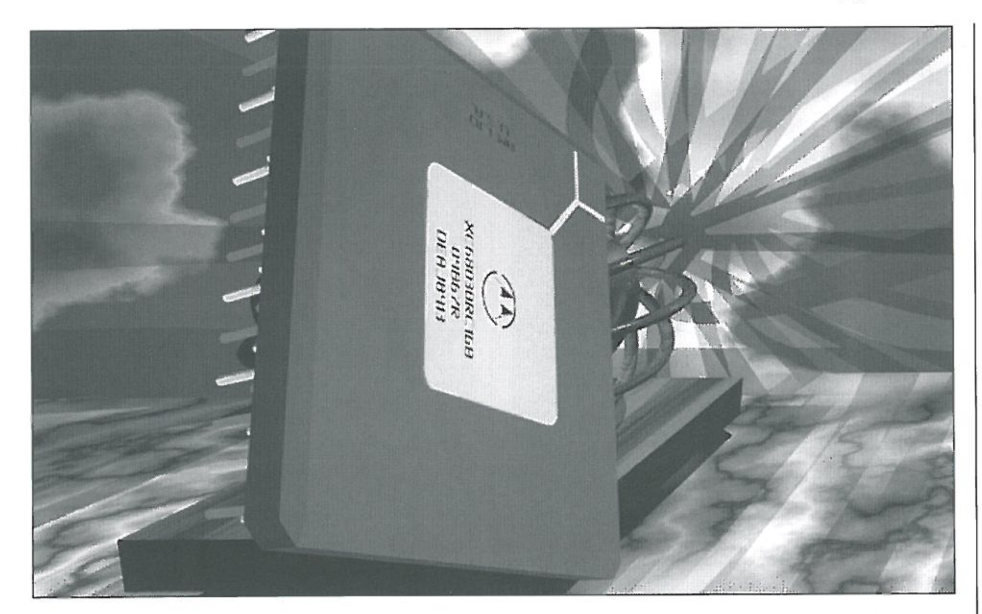
- Reconversión del tipo de color original (de YCbCr a RGB). Si se trata de una imagen en escala de grises, este paso se omite. Aplicación de un posible ajuste gamma.

- Con los datos raw se creará el fichero de salida deseado.

EJEMPLO PRÁCTICO

En el disco que acompaña a la revista, se han incluido dos programas





para el manejo de imágenes JPEG. Uno para comprimir al formato JPEG (cjpeg.exe) y otro para descomprimir un fichero JPEG a un formato convencional (djpeg.exe).

Admiten en sus parámetros de entrada varios modificadores, además de los nombres de los ficheros de entrada y de salida. Los formatos de ficheros gráficos permitidos son: PPM (fichero PBMPLUS color), PGM (fichero PBMPLUS de escala de grises), GIF y Targa.

La mayoría de los parámetros de entrada pueden teclearse abreviados como -gray o -gr en vez de -grayscale.

CJPEG

El tipo del fichero de entrada es reconocido automáticamente.

Los parámetros de entrada son los siguientes:

-quality N: Modifica el tamaño de las tablas de cuantización para ajustar la calidad de la imagen. El valor 0 es el de peor calidad, y el 100 el que crea una calidad mejor, aunque el nivel de compresión degenere enormemente.

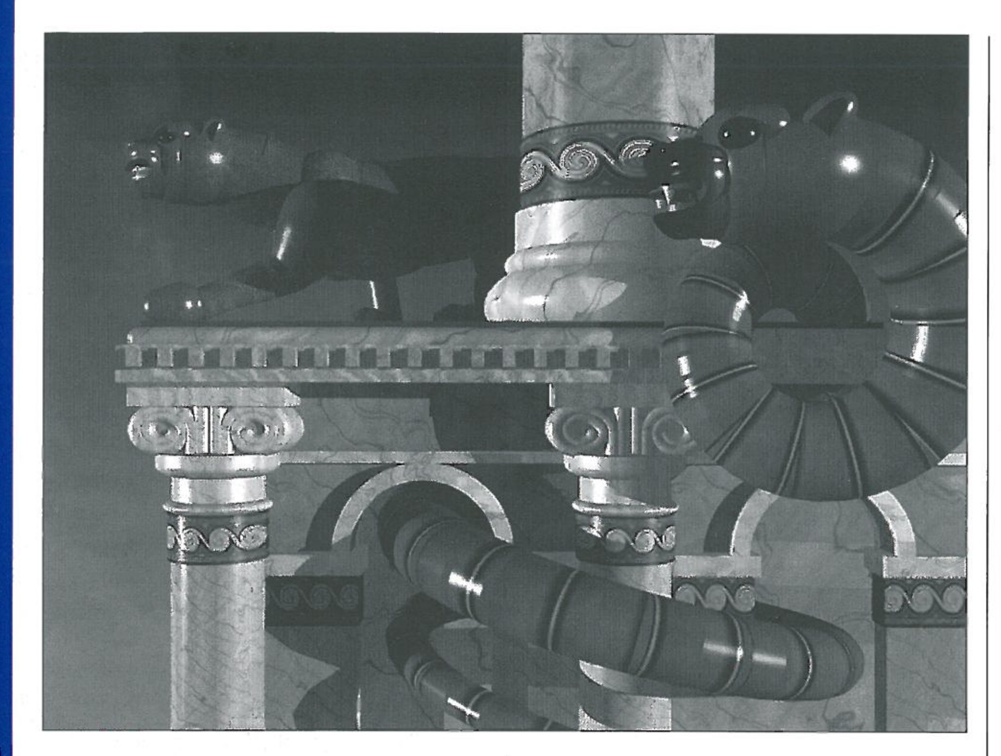
-grayscale: Genera un fichero de escala de grises.

-optimize: Realiza una optimización de codificación entrópica. Sin este parámetro se utiliza una codificación normal. Sirve para generar un fichero JPEG un poco más pequeño.

-targa: Confirma que el fichero de entrada está en formato Targa.

También están disponibles algunos parámetros para modificar valores sofisticados:

-maxmemory N: Indica el límite de memoria a utilizar en el proceso de compresión. El valor indica Kb o Mb



si se añade una "m" al final del número.

-restart N: Escribe una marca de reinicio cada N filas de MCU o cada N bloques MCU, si la letra "b" es añadida al número.

-smooth N: "Suaviza" la imagen de entrada para eliminar posible "ruido" en la imagen. Los valores posibles van de 1 a 100 (un 0 indica que no se realice la operación). Un valor entre 10 a 50 crea un fichero JPEG más pequeño y suele mejorar su aspecto.

-verbose: Activa la visualización de la información de depurado en la conversión.

-qtables fichero: Utiliza las tablas de cuantización almacenadas en el fichero especificado.

Este debe contener de una a cuatro tablas (de 64 valores cada una) en formato texto. Admite líneas de comentarios encabezadas con la letra "#".

-sample HxV[,...]: Fija los factores de muestreo JPEG. El valor por defecto es equivalente a -sample 2x2.

DJPEG

Sus parámetros de entrada son:

-colors N o -quantize N: Reduce la imagen a N colores en el fichero de salida.

-gif: Selecciona el formato de salida GIF (-colors 256 es asumido si no se especifica un número menor de colores).

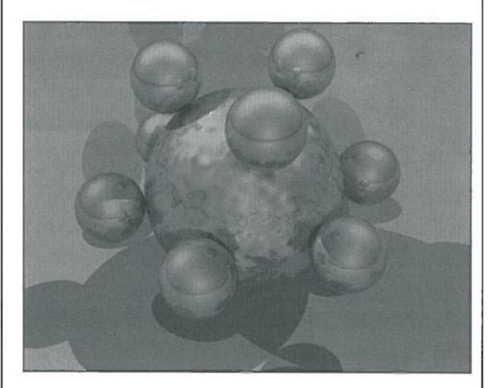
-pnm: Selecciona el formato de salida PBMPLUS (PPM/PGM). Es el formato por defecto.

-targa: Selecciona el formato de salida Targa.

Los parámetros avanzados son:

-blocksmooth: Realiza un smoothing a nivel de bloque.

-grayscale: Obliga a una salida en escala de grises.



-maxmemory N: Indica el límite de memoria a utilizar en el proceso de compresión. El valor indica Kb o Mb si se añade una "m" al final del número.

-nodither: No utiliza dithering en la cuantización del color. Por defecto, se usa uno del tipo Floyd-Steinberg.

-onepass: Utiliza solo una pasada en la cuantización del color, en vez de las dos habituales.

-verbose: Activa la visualización de la información de depurado en la conversión.

BIBLIOGRAFÍA

Discrete Cosine Transform—-Algorithms, Advantages, Applications
K.R. Rao y P. Yip (Academic Press, Inc, London, 1990)

The Independent JPEG Group's JPEG Software
Versión 4 del 10 de Diciembre de 1992

JPEG Still Image Compression Standard
William B. Pennebaker y Joan L. Mitchell
Van Nostrand Reinhold, 1993

The JPEG Still Picture Compression Standard
Wallace, Gregory K.
Boletín ACM de Abril del 1991 (vol. 34 no. 4)

The Data Compression Book
Mark Nelson
M&T Books (Redwood City, CA), 1991

Practical Fast 1-D DCT Algorithms with 11 Multiplications
Proc. Int'l. Conf. on Acoustics, C. Loeffler, A. Ligtenberg y G. Moschytz
Speech, and Signal Processing 1989 (ICASSP '89)

GLOSARIO

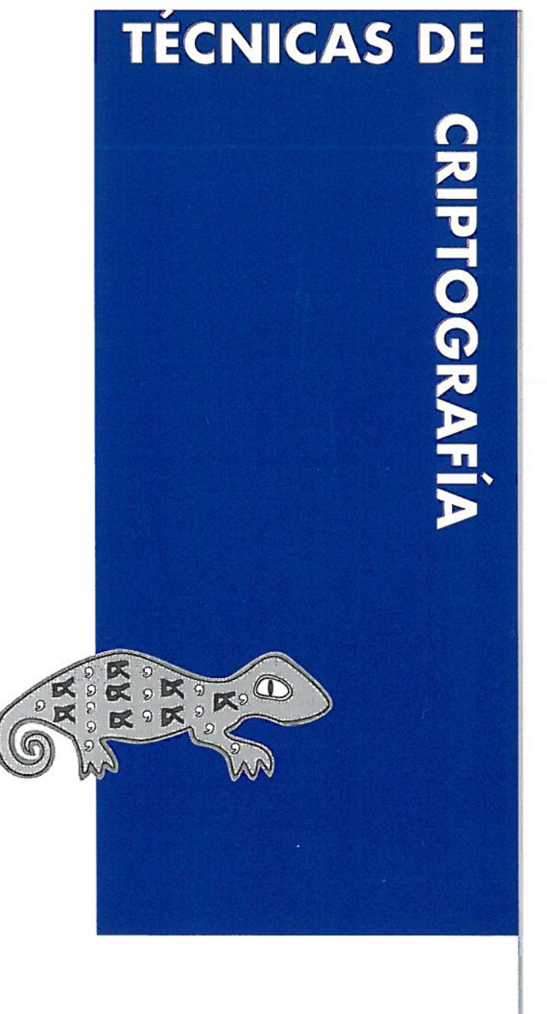
Componente: Significa un canal de color, como por ejemplo, rojo, verde, azul o luminancia.

DCT: Discrete Cosine Transform (Transformación de Coseno Separada), es el algoritmo que se encarga de comprimir los datos de la imagen.

IDCT: Inverse Discrete Cosine Transform (Transformación Inversa de Coseno Separada) , es el algoritmo que se encarga de descomprimir los datos de la imagen.

MCU: Minimum Code Unit (Unidad Mínima Codificada), se trata de cada uno de los bloques en los que se divide la imagen para la compresión/descompresión (habitualmente 8x8 pixels).

Sample: Es el valor del componente de un pixel, o lo que es lo mismo, un número en los datos de la imagen. ■



LICENCIA PARA ENGAÑAR

Rafael A. Cazorla

Desde siempre, el hombre ha buscado el poder, el dominio sobre el resto de los mortales, y para conseguirlo no basta con ser más fuerte, sino que hace falta información (mapas, detalles sobre el enemigo, etc.) y que ésta tenga un uso restringido a un grupo muy determinado de personas. Como suele ocurrir con muchos de los avances tecnológicos que realiza el hombre, la criptografía fue desarrollada para realizar esa tarea en la guerra.

Su misión consiste en transmitir mensajes secretos, de modo que si el mensajero es capturado, y con él, el papiro con el texto, resultase prácticamente imposible de descifrar por el enemigo. Los primeros usos conocidos de la criptografía se remontan a Julio Cesar, y fue a partir de entonces muy utilizada. En el Renacimiento, las relaciones diplomáticas impusieron este procedimiento, y en cada cancillería había secretarios de cifra, encargados de transmitir o descifrar los despachos. Los métodos más comunes, hasta prácticamente este siglo eran de transposición (alteración del orden de las letras) y de sustitución (cambios de unos elementos por otros). La transposición puede ser simple, doble y con clave. La simple consiste, en colocar las letras del texto cifrado en un cuadrado o rectángulo, dividido en tantas líneas de casillas como sean necesarias, para colocar una letra del texto en cada una. Dividiendo el número de letras del texto, por el número clave, se averiguan las líneas que hay que construir; el resto de la división señalará el sobrante de cuadrados en blanco.

Estos métodos primitivos no presentan una elevada complejidad para ser descifrados, ya que en casi todos se presenta una correspondencia biyectiva, entre letras y símbolos (ejemplo: será "a", igual a "b",...) por lo tanto si el texto cifrado es de cierta longitud, haciendo un estudio del número de veces que aparece cada símbolo, de la manera de están agrupados, etc. sólo es cuestión de tiempo conseguir descifrarlo (método que emplea Edgar Allan Poe en su novela "El escarabajo de oro").

A continuación se explicarán dos métodos muy empleados en informática, y en todo aquello que se refiere a comunicaciones, los protocolos:

- Código N x N (matricial)
- Código R.S.A.

CÓDIGOS N x N

Lo primero es asignar del 1 al 28 (se utilizará aquí también la "ñ " ... siendo en vez de Z28 en castellano Z29) cada letra del alfabeto siendo el espacio entre palabras el numero 0, de esta manera se obtiene, utilizando aritmética modular, o aritmética en Zn, la biyección entre el alfabeto y Z29. También se podrían utilizar los 256 caracteres ASCII siendo la biyección con Z256.

En este caso será :

"_" - 0, a- 1, b - 2, c - 3, ...n-14, ñ-15,... ,y- 26, z - 28.

A continuación agruparemos los números en matrices, columna en este

"SÓLO PROGRAMADORES", N=2

$$\begin{Bmatrix}20\\16\end{Bmatrix},\begin{Bmatrix}12\\16\end{Bmatrix},\begin{Bmatrix}0\\17\end{Bmatrix},\begin{Bmatrix}19\\16\end{Bmatrix},\begin{Bmatrix}7\\19\end{Bmatrix}$$

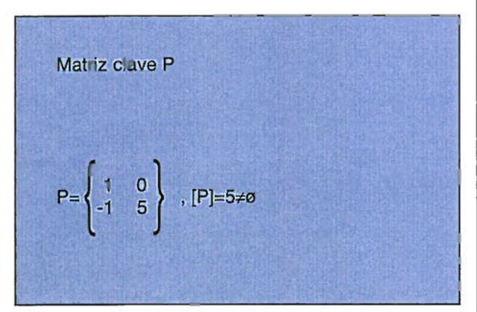
$$\begin{Bmatrix}1\\13\end{Bmatrix},\begin{Bmatrix}1\\4\end{Bmatrix},\begin{Bmatrix}16\\19\end{Bmatrix},\begin{Bmatrix}20\\0\end{Bmatrix}$$

Cuadro 1: Texto a decodificar.

¡TORA TORA TORA! Quien haya visto esta película, recordará que antes del ataque a Pearl Harbor, los americanos habían interceptado una transmisión japonesa, que no consiguieron descifrar, hasta que ya fue inevitable el bombardeo. Esta tardanza se debió, a que el mensaje estaba codificado criptográficamente.

caso, de orden NxN, siendo esta N una clave aleatoria.

El siguiente paso para encriptar es multiplicar cada matriz o vector por una matriz P que posea inverso en Z29, esto es, que su determinante sea un número invertible en Z29. Por ejemplo:



Cuadro 2: matriz clave P.

Para hallar la matriz X por la que se multiplicará para codificar, se calcula la adjunta de la traspuesta de p, y se la multiplica por el inverso del determinante de p, y ésta se pasa a módulo 29.

Matriz X y $[P]^{-1}$

$$X=\begin{Bmatrix} 1 & 0 \\ 6 & 6 \end{Bmatrix}, \quad [P]^{-1}=6$$

Cuadro 3: Matriz X e inverso del det(p).

Ahora se procede a la encriptación multiplicando X por todas las matrices, codificando el resultado en módulo 29, siendo el resultado:

Resultado de la codificación NxN

$$X\bullet\begin{Bmatrix} 20 \\ 16 \end{Bmatrix}=\begin{Bmatrix} 20 \\ 216 \end{Bmatrix}\xrightarrow{Z_{29}}\begin{Bmatrix} 20 \\ 13 \end{Bmatrix} \quad : s, l$$

$$X\bullet\begin{Bmatrix} 12 \\ 16 \end{Bmatrix}=\begin{Bmatrix} 12 \\ 168 \end{Bmatrix}\xrightarrow{Z_{29}}\begin{Bmatrix} 12 \\ 23 \end{Bmatrix} \quad : l, v$$

$$X\bullet\begin{Bmatrix} 0 \\ 17 \end{Bmatrix}=\begin{Bmatrix} 0 \\ 102 \end{Bmatrix}\longrightarrow\begin{Bmatrix} 0 \\ 15 \end{Bmatrix} \quad : \sqcup, ñ$$

$$X\bullet\begin{Bmatrix} 19 \\ 16 \end{Bmatrix}= \qquad \begin{Bmatrix} 19 \\ 7 \end{Bmatrix} \quad : r, g$$

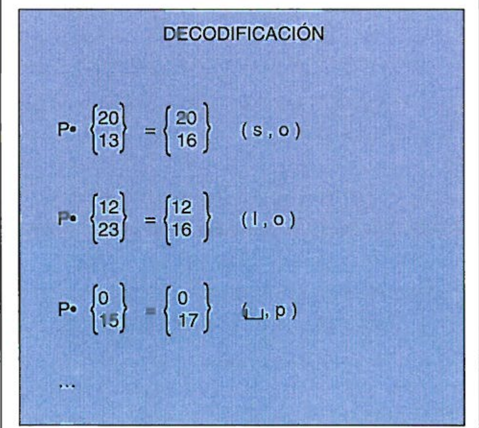
... ...

Cuadro 4: Resultado de la codificación NxN.

SMLV_ÑRGGKAXAAOGSD.

Con lo que se comprueba que ya no existe una correspondencia biyectiva puesto que "smlv" representa la palabra "SOLO" donde las "os" ya no son representadas por el mismo número, ni letra.

Para decodificarlo, simplemente habrá que realizar el proceso inverso. Primero se agruparán las letras en matrices de NxN, cada letra se representará con su correspondiente en Z29, y se multiplicarán estas matrices por p, obteniéndose el texto original:



Cuadro 5: Decodificación.

texto original: SOLO PROGRAMADORES

Este método, incluso con esta representación tan sencilla de 2 x 2, alcanza un índice de seguridad muy elevado, ya que si de las 29(4)= 707.281 matrices posibles de orden 2x2 en Z29 son invertibles un gran numero de ellas, obligaría a probar por todas ellas hasta dar con P si esta es desconocida, y como se puede comprobar, esto requiere mucho tiempo. Aunque también tiene su inconveniente: la clave P debe ser conocida por quien envía el texto y por quien lo recibe.

CÓDIGOS RSA

Este método fue desarrollado en 1977 por R.L. Rivest, A. Shamir y L. Adleman, cuyas siglas representan el nombre R.S.A.. Crearon un código, cuyas claves no tenían que ser conocidas por el receptor ni por el emisor. Este tipo de codificación se le denomina "de clave publica", y están basados en la teoría de la divisibilidad.

Actualmente es considerado como el sistema criptográfico más fiable para la protección de la información. Su uso por tanto, es vital en aquellas circunstancias donde es posible la intrusión de un sneaker (fisgón) cualquiera, que ande sediento de datos privilegiados. Los expertos en seguridad tienen un lema: "Andar un paso por delante de los ladrones", y ¿qué mejor arma, que evitar que puedan utilizar lo que hayan conseguido robar?. Podemos encontrar datos protegidos de esta manera en las tarjetas inteligentes (telefónica, tarjetas de crédito), bases de datos, incluso en la red de ordenadores instalada en su oficina, donde los datos antes de ser lanzados de un ordenador a otro se codifican para evitar escuchas indebidas.

SU CODIFICACIÓN

El primer paso a seguir es trasladar el texto a su traducción numérica en Z28 (o Z29, el método es el mismo). Después se reagrupa el texto en bloques de una misma longitud "r". Se ha de tener en cuenta que cada letra será representada por dos dígitos, por tanto, a la letra "A" le corresponde en Z28 el número "01". A cada uno de los bloques resultantes se les codificara individualmente, por lo que se les denominará "palabra".

Ejemplo:

Texto a codificar: "HOLA A TODOS".

Agrupación inicial (r=4): (HOLA) (A T) (ODOS).

Paso a Z28: 08161201 00010021 1604162066.

El espía imaginario creado, ha de mandar el texto codificado a su base, por lo tanto ha de crear su clave publica, que consta de dos cifras "q" y "s". La norma para la elección de "q" es que sea primo con las palabras del texto, de manera que resulte un código realmente seguro, por lo que se debe tomar el producto, de dos primos de más de 2r+1 dígitos cada uno. Para hallar "s" se debe tomar un número primo con f(q).

f(q) es la función de Euler y representa la cifra de números coprimos de

"q". Puede tomar dos formas dependiendo de q:

Si q es primo: f(q) = q1

Si no: f(q)= q[1(1/p1)] x [1-(1/p2)] x [1(1/p3)]...

siendo p1,p2,p3... los factores primos de q.)

La forma de encriptar cada palabra consiste en elevar cada una a "s" devolviendo el resultado en módulo (q). Tomando como ejemplo la palabra (9171302), como "q" a (3524084471) y s=5 (al ser f(q) un número primo con 5) se obtiene:

PALABRA CODIFICADA: 9171302 e5 modulo (q)= 2839270855.

DECODIFICACIÓN

La decodificación consiste en codificar de nuevo las palabras que previamente habían sido codificadas, pero esta vez se utilizará "t" en lugar de "s" que es su inverso en modulo f(q).

Si q=3524084471 y s=5 entonces t= 740793149, por lo tanto, la palabra decodificada será :

$$(2839270855)^E \text{ mod } q$$

Cuadro 6: Palabra decodificada.

Este método aunque es fácil de realizar presenta dos problemas en su cálculo:

1. El número t posiblemente sea muy elevado, lo que acarrea que cuando se decodifique, al elevar una palabra a t se necesiten fácilmente potentes ordenadores para acelerar la encriptación de textos largos, y su posterior desencriptación. Esto evidentemente tiene un gran costo en operaciones, y sólo se debe utilizar cuando no haya alternativa. Existe un sistema que simplifica el cálculo, que proporciona a la vez cifras más manejables, puesto que todas serán menores que "q". Éste consiste en pasar t a binario natural, una vez traducido se intercala entre cada dígito la letra "C" se sustituyen los "1" por "M" finalizando con la eliminación de los "0". Las "M" significan multiplicar por la palabra y las "C" es elevar al cuadrado. Por ejemplo:

t= 3 mod(101) -> 11 en binario -> 1 C 1 -> M C M ->
-> M C M

Por lo tanto si la palabra codificada es 061 se traduciría a:

061 e3 mod(101) -> 1->(M) 61(101) ->(C) 721=85(101) ->(M) 5185= 34(101).

Si se sabía que r=2, y que el alfabeto utilizado era: "_, E, M, P, C, O, R, S " por lo tanto de Z7, sólo se necesita separar el 34 en dos números: 3 y 4 que corresponden a la palabra - PC -. Con este método se reduce el número de operaciones a un orden de $\log_2 t$.

2. El otro problema se trata de la dificultad en hallar los factores primos de "q" que son necesarios para conocer t. Pero en esto precisamente radica la seguridad del RSA, puesto que resulta muy complicado saber si un número de 10 o 12 cifras es primo o no, es más, existen algoritmos que permiten buscar números con la garantía, de que ningún otro algoritmo que se utilice para localizarlo lo identificará como primo. Sólo podrá factorizar q para encontrar f(q) quien lo calculó, por lo tanto es imposible su decodificación.

COMUNICACIÓN DEL MENSAJE

El primer paso es determinar "r", entre el que envía el mensaje (A) y el que lo recibe (B). Para demostrar que su funcionamiento es correcto existirá también un sujeto (S), que intentará descifrar el mensaje codificado de A. Los tres personajes (A,B,C) construirán, q(A), q(B), y q(C) respectivamente, para posteriormente hacer públicas las claves (q,s) de cada uno.

Si A quiere enviar un mensaje (M) a B realizará el siguiente proceso: codifica (M,(q(A),s(A)) con la clave (q(A), t(A)), que es la que define quien lo envía, para más tarde volverla a encriptar con la clave (q(B), s(B)).

$$(M,q_A,S_A) \xrightarrow{q_A\ t_A} (M^1,q^1,S^1) \xrightarrow{q_B\ S_S} (X,q,S)$$

Esquema 1: Codificación de A a B.

B al recibir el mensaje de A (X, q, s) ha de volver a codificar dos veces el mensaje encriptado X, primero usando la clave propia (q(B),t(B)) y después usando la pública de A (q(A), s(A)), con lo que consigue obtener el mensaje original enviado de A.

$$(E,q,s) \xrightarrow{q_B\ t_B} (M^1,q^1,S^1) \xrightarrow{q_A\ S_A} (M,q_A,s_A)$$

Esquema 2: Decodificación de B a A.

Si C hubiese interceptado el mensaje de A, y quisiese engañar a B con un falso mensaje de A, sería imposible puesto que C desconoce t(A), y aunque se inventara un t(X) parecido al de A, B descubriría el engaño al no obtener q(A) ni s(A) al decodificar el mensaje.

$$(M_C,q_A,S_A) \xrightarrow{q_A\ t^1_A} (M^1,q^1,S^1) \xrightarrow{q_B\ S_B} (E,q,s)$$

$$(E,q,S) \xrightarrow{q_B\ t_B} (M^1,q^1,S^1) \xrightarrow{q_A\ S_A} (M^1_C,q_x,s_x)$$

Esquema 3: Decodificación ilegal.

Como se habrá podido entender, la seguridad de este método de encriptación es directamente proporcional, a la capacidad de generar grandes números pseudoprimos por la máquina que se utilice, por ello los creadores del RSA garantizaron, que si se codificaba correctamente, los posibles intentos de descifrar el mensaje consumirían al menos dos generaciones de espías, incluso utilizando potentes ordenadores.

Los programas que se adjuntan son ejemplos de como funcionarían básicamente los encriptadores, que utilicen los métodos RSA y 1x1. Están codificados en C. y utilizan las librerías del compilador Turbo C de Borland versión 2, por lo que resultarán fácilmente portables a otras versiones. ■

EL CONOCIMIENTO AL PODER

Juan José Samper

Antes de la aparición del ordenador, el hombre ya se preguntaba si se le arrebataría el privilegio de razonar y pensar. En la actualidad existe un campo dentro de la inteligencia artificial al que se le atribuye esa facultad: el de los sistemas expertos.

Estos sistemas permiten la creación de máquinas que razonan como el hombre, restringiéndose a un espacio de conocimientos limitado. En teoría pueden razonar siguiendo los pasos que seguiría un experto humano (médico, analista, empresario, etc.) para resolver un problema concreto. Este tipo de modelos de conocimiento por ordenador ofrece un extenso campo de posibilidades, en resolución de problemas y en aprendizaje. Su uso se extenderá ampliamente en el futuro, debido a su importante impacto sobre los negocios y la industria.

El objetivo de este curso es enseñar de forma práctica el funcionamiento y construcción de un sistema experto. A lo largo de la serie de artículos se demostrará que los sistemas expertos son útiles y prácticos, y que, además, son realizables.

También se analizará la problemática de la adquisición y representación del conocimiento, así como los métodos para tratar la incertidumbre.

Cada mes se confeccionará una parte de un pequeño sistema experto que el lector podrá adaptar a sus necesidades.

¿QUÉ ES UN SISTEMA EXPERTO?

Los sistemas expertos se pueden considerar como el primer producto verdaderamente operacional de la inteligencia artificial. Son programas de ordenador diseñados para actuar como un especialista humano en un dominio particular o área de conocimiento. En este sentido, pueden considerarse como intermediarios entre el experto humano, que transmite su conocimiento al sistema, y el usuario que lo utiliza para resolver un problema con la eficacia del especialista. El sistema experto utilizará para ello el conocimiento que tenga almacenado y algunos métodos de inferencia.

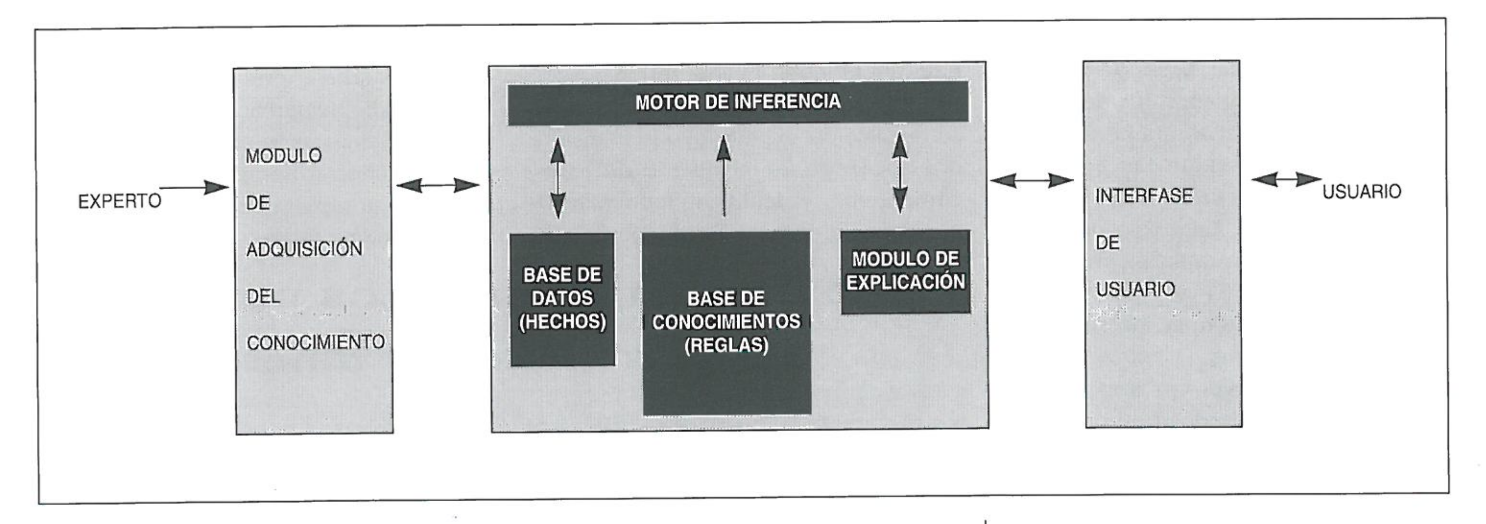
El sistema experto actúa como un especialista humano en un área concreta de conocimiento.

A la vez, el usuario puede aprender observando el comportamiento del sistema. Es decir, los sistemas expertos se pueden considerar simultáneamente como un medio de ejecución y transmisión del conocimiento.

Lo que se intenta, de esta manera, es representar los mecanismos heurísticos que intervienen en un proceso de descubrimiento. Estos mecanismos forman ese conocimiento difícil de expresar, que permite a los expertos humanos ser eficaces, calculando lo menos posible. Los sistemas expertos contienen ese "saber hacer".

La característica fundamental de un sistema experto es, que separa los conocimientos almacenados (base de conocimiento), del programa que los controla (motor de inferencia). Los da-

Los sistemas expertos son programas que reproducen el proceso intelectual de un experto humano en un campo particular, pudiendo mejorar su productividad, ahorrar tiempo y dinero, conservar sus valiosos conocimientos y difundirlos más fácilmente.



tos propios de un determinado problema se almacenan en una base de datos aparte (base de hechos).

Una característica adicional deseable, y a veces fundamental, es que el sistema sea capaz de justificar su propia línea de razonamiento, de forma inteligible por el usuario.

Los sistemas expertos siguen una filosofía diferente a los programas clásicos. Esto queda reflejado en la tabla 1, que resume las diferencias entre ambos tipos de procesamiento.

El usuario aprende al observar el comportamiento del sistema experto

LA RECIENTE HISTORIA DE LOS SISTEMAS EXPERTOS

Los sistemas expertos proceden inicialmente de la inteligencia artificial a mediados de los años sesenta. En ese período, se creía que bastaban unas pocas leyes de razonamiento, junto con potentes ordenadores para producir resultados brillantes. Un intento en ese sentido fue el llevado a cabo por los investigadores Alan Newell y Herbert Simon, que desarrollaron un programa denominado GPS (General Problem Solver; solucionador general de problemas). Podrá trabajar con criptoaritmética, con las torres de Hanoi y con otros problemas similares. Lo que no podrá hacer el GPS, era resolver problemas del mundo real, tales como un diagnóstico médico.

Algunos investigadores decidieron entonces cambiar por completo el enfoque del problema, restringiendo su ambición a un dominio específico, e intentando simular el razonamiento de un experto humano. En vez de dedicarse a computerizar la inteligencia general, se centraron en dominios de conocimiento muy concretos. De esta manera nacieron los sistemas expertos.

A partir de 1965, un equipo dirigido por Edward Feigenbaum, comenzó a desarrollar sistemas expertos utilizando bases de conocimiento definidas minuciosamente.

En 1967 se construye DENDRAL, que se considera como el primer sistema experto. Se utilizaba para identificar estructuras qu'micas moleculares a partir de su análisis espectrográfico.

Entre 1970 y 1980 se desarrolló MYCIN para consulta y diagnóstico de infecciones de la sangre. Este sistema introdujo nuevas características: utilización de conocimiento impreciso para razonar y posibilidad de explicar el proceso de razonamiento. Lo más importante es que funcionaba de manera correcta, dando conclusiones análogas a las que un ser humano dar'a tras largos años de experiencia. En MYCIN aparecen claramente diferenciados motor de inferencia y base de conocimientos. Al separar esas dos partes, se puede considerar el motor de inferencias aisladamente. Esto da como resultado un sistema vacío o shell (concha). Así surgió EMYCIN (MYCIN Esencial) con el que se construyó SACON, utilizado para estructuras de ingenier'a, PUFF para estudiar la función pulmonar y GUIDON para elegir tratamientos terapéuticos.

En esa época se desarrollaron también: HERSAY, que intentaba identificar la palabra hablada, y PROSPECTOR, utilizado para hallar yacimientos de minerales. De este último derivó el shell KAS (Knowledge Adquisition System).

A partir de 1980 se ponen de moda los sistemas expertos, numerosas empresas de alta tecnología investigan en este área de la inteligencia artificial, desarrollando sistemas expertos para su comercialización. Se llega a la conclusión de que el éxito de un sistema experto depende casi exclusivamente de la calidad de su base de conocimiento. El inconveniente es que codificar la pericia de un experto humano puede resultar difícil, largo y laborioso.

Un ejemplo de sistema experto moderno es CASHVALUE, que evalúa proyectos de inversión y VATIA, que asesora acerca del impuesto sobre el valor añadido o I.V.A.

USOS DE UN SISTEMA EXPERTO

Un sistema experto es muy eficaz cuando tiene que analizar una gran cantidad de información, interpretándola y proporcionando una recomendación a partir de la misma. Un ejemplo es el análisis financiero, donde se estudian las oportunidades de inversión, dependiendo de los datos financieros de un cliente y de sus propósitos.

Para detectar y reparar fallos en equipos electrónicos, se utilizan los sis-

temas expertos de diagnóstico y depuración, que formulan listas de preguntas con las que obtienen los datos necesarios para llegar a una conclusión. Entonces recomiendan las acciones adecuadas para corregir los problemas descubiertos. Este tipo de sistemas se utilizan también en medicina (ej. MYCIN y PUFF), y para localizar problemas en sistemas informáticos grandes y complejos.

Los sistemas expertos son buenos para predecir resultados futuros a partir del conocimiento que tienen. Los sistemas meteorológicos y de inversión en bolsa son ejemplos de utilización en este sentido. El sistema PROSPECTOR es de este tipo.

La planificación es la secuencia de acciones necesaria para lograr una meta. Conseguir una buena planificación a largo plazo es muy difícil. Por ello, se usan sistemas expertos para gestionar proyectos de desarrollo, planes de producción de fábricas, estrategia militar y configuración de complejos sistemas informáticos, entre otros.

Cuando se necesita controlar un proceso tomando decisiones como respuesta a su estado, y no existe una solución algorítmica adecuada, es necesario usar un sistema experto. Este campo comprende el supervisar fábricas automatizadas, factorías químicas o centrales nucleares. Estos sistemas son extraordinariamente críticos porque normalmente tienen que trabajar en tiempo real.

El diseño requiere una enorme cantidad de conocimientos debido a que hay que tener en cuenta muchas especificaciones y restricciones. En este caso, el sistema experto ayuda al diseñador a completar el diseño de forma competente, y dentro de los límites de costes y de tiempo. Se diseñan circuitos electrónicos, integrados, tarjetas de circuito impreso, estructuras arquitectónicas, coches, piezas mecánicas, etc.

Por último, un sistema experto puede evaluar el nivel de conocimientos y comprensión de un estudiante, y ajustar el proceso de aprendizaje de acuerdo con sus necesidades.

En la tabla 2 se muestran los modelos funcionales de los sistemas expertos, junto al tipo de problema que intentan resolver y algunos de los usos concretos a que se destinan.

¿SISTEMA EXPERTO, SÍ O NO?

El acceso al conocimiento y al juicio de un experto es extremadamente valioso en muchas ocasiones (prospecciones petrolíferas, manejo de valores bursátiles, diagnóstico de enfermedades, etc.), sin embargo, en la mayoría de los campos de actividad existen más problemas por resolver que expertos para resolverlos. Para solucionar este desequilibrio es necesario utilizar un sistema experto. En general, actuará como ayudante para los expertos humanos y como consultor cuando no se tiene otro acceso a la experiencia.

Un sistema experto, además, mejora la productividad al resolver y decidir los problemas más rápidamente. Esto permite ahorrar tiempo y dinero. A veces sin esa rapidez las soluciones obtenidas serían inútiles.

Los sistemas expertos pueden ahorrar tiempo y dinero

Los valiosos conocimientos de un especialista se guardan y se difunden, de forma que, no se pierden aunque desaparezca el especialista. En los sistemas expertos se guarda la esencia de los problemas que se intenta resolver y se programa cómo aplicar los conocimientos para su resolución. Ayudan a entender cómo se aplican los conocimientos para resolver un problema. Esto es útil porque normalmente el especialista da por ciertos sus conocimientos y no analiza cómo los aplica.

Se pueden utilizar personas no especializadas para resolver problemas. Además si una persona utiliza regularmente un sistema experto aprenderá de él, y se aproximará a la capacidad del especialista.

Con un sistema experto se obtienen soluciones más fiables gracias al tratamiento automático de los datos, y más contrastadas, debido a que se suele tener informatizado el conocimiento de varios expertos.

Debido a la separación entre la base de conocimiento y el mecanismo de inferencia, los sistemas expertos tienen gran flexibilidad, lo que se traduce en una mejor modularidad, modificabilidad y legibilidad del conocimiento.

Otra ventaja es que este tipo de sistemas pueden utilizar razonamiento aproximado para hacer deducciones y que pueden resolver problemas sin solución algorítmica.

Los sistemas expertos también tienen inconvenientes. El conocimiento humano es complejo de extraer y, a veces, es problemático representarlo. Si un problema sobrepasa la competencia de un sistema experto, sus prestaciones se degradan de forma notable. Además, las estrategias de razonamiento de los motores de inferencia,

SISTEMA CLÁSICO	SISTEMA EXPERTO
Conocimiento y procesamiento combinados en un programa	Base de conocimiento separada del mecanismo de procesamiento
No contiene errores	Puede contener errores
No da explicaciones, los datos sólo se usan o escriben	Una parte del sistema experto la forma el módulo de explicación
Los cambios son tediosos	Los cambios en las reglas son fáciles
El sistema sólo opera completo	El sistema puede funcionar con pocas reglas
Se ejecuta paso a paso	La ejecución usa heur'stica y lógica
Necesita información completa para operar	Puede operar con información incompleta
Representa y usa datos	Representa y usa conocimiento

Tabla 1. Comparación entre un sistema clásico de procesamiento y un sistema experto.

suelen estar programadas proceduralmente y se adaptan mal a las circunstancias. Están limitados para tratar problemas con información incompleta.

Un experto humano no estudia progresivamente una hipótesis, sino que decide de inmediato cuando se enfrenta a una situación análoga a otra ocurrida en el pasado. Los sistemas expertos no utilizan este razonamiento por analogía.

Los costes y duración del desarrollo de un sistema experto son bastante considerables (aunque se suelen amortizar rápidamente), y su campo de aplicación actual es restringido y específico.

Finalmente, hay que tener en cuenta los problemas sociales que acarrean, al ser susceptibles de influir en la estructura y número de empleos.

ARQUITECTURA Y FUNCIONAMIENTO DE UN SISTEMA EXPERTO

No existe una estructura de sistema experto común. Sin embargo, la mayoría de los sistemas expertos tienen unos componentes básicos: base de conocimientos, motor de inferencia, base de datos, e interfaz con el usuario. Muchos tienen, además, un módulo de explicación y un módulo de adquisición del conocimiento. La figura 1 muestra la estructura de un sistema experto ideal.

La base de conocimientos contiene el conocimiento especializado extraído del experto en el dominio. Es decir, contiene conocimiento general sobre el dominio en el que se trabaja. El método más común para representar el conocimiento es mediante reglas de producción. El dominio de conocimiento representado se divide, pues, en pequeñas fracciones de conocimiento o reglas SI . . . ENTONCES ...

Cada regla constará de una parte denominada condición y de una parte denominada acción, y tendrá la forma:

SI condición ENTONCES acción

Como ejemplo se puede considerar la siguiente regla médica:

SI el termómetro marca 39º
Y el termómetro funciona correctamente
ENTONCES el paciente tiene fiebre

Una característica muy importante es, que la base de conocimientos es independiente del mecanismo de inferencia que se utiliza para resolver los problemas. De esta forma, cuando los conocimientos almacenados se han quedado obsoletos, o cuando se dispone de nuevos conocimientos, es relativamente fácil añadir reglas nuevas, eliminar las antiguas, o corregir errores en las existentes. No es necesario reprogramar todo el sistema experto.

La característica fundamental de un sistema experto es la separación entre conocimiento e inferencia

Las reglas suelen almacenarse en alguna secuencia jerárquica lógica, pero esto no es estrictamente necesario. Se pueden tener en cualquier secuencia, y el motor de inferencia las usará en el orden adecuado que necesite para resolver un problema.

Una base de conocimientos muy ingenua, para identificar vehículos, podría ser la siguiente:

Regla 1: SI tiene 2 ruedas
Y utiliza motor
ENTONCES es una motocicleta
Regla 2: SI tiene 2 ruedas
Y es movido por el hombre
ENTONCES es una bicicleta
Regla 3: SI tiene 4 ruedas
Y utiliza motor
Y pesa menos de 3500 Kg.
ENTONCES es un coche

Existen reglas de producción que no pertenecen al dominio del problema. Estas reglas se llaman meta-reglas (reglas sobre otras reglas), y su función es indicar, bajo qué condiciones deben considerarse unas reglas en vez de otras. Un ejemplo de meta-regla es:

SI hay reglas que usan materias baratas
Y hay reglas que usan materias caras
ENTONCES usar antes las primeras que las segundas

La base de datos o base de hechos es una parte de la memoria del ordenador, que se utiliza para almacenar los datos recibidos inicialmente para la resolución de un problema. Contiene conocimiento sobre el caso concreto en que se trabaja. También se registrarán en ella las conclusiones intermedias y los datos generados en el proceso de inferencia. Al memorizar todos los resultados intermedios, conserva el vestigio de los razonamientos efectuados; por lo tanto, se puede utilizar explicar las deducciones y el comportamiento del sistema.

El motor de inferencias, es un programa que controla el proceso de razonamiento que seguirá el sistema experto. Utilizando los datos que se le suministran, recorre la base de conocimientos para alcanzar una solución. La estrategia de control puede ser de encadenamiento progresivo o de encadenamiento regresivo.

En el primer caso se comienza con los hechos disponibles en la base de datos, y se buscan reglas que satisfagan esos datos, es decir, reglas que verifiquen la parte "SI". Normalmente, el sistema sigue los siguientes pasos:

1. Evaluar las condiciones de todas las reglas respecto a la base de datos, identificando el conjunto de reglas que se pueden aplicar (aquellas que satisfacen su parte condición)
2. Si no se puede aplicar ninguna regla, se termina sin éxito; en caso contrario se elige cualquiera de las reglas aplicables y se ejecuta su parte acción (esto último genera nuevos hechos que se a–aden a la base de datos)
3. Si se llega al objetivo, se ha resuelto el problema; en caso contrario, se vuelve al paso 1

A este enfoque se le llama también guiado por datos, porque es el estado de la base de datos el que identifica las reglas que se pueden aplicar. Cuando

se utiliza este método, el usuario comenzará introduciendo datos del problema en la base de datos del sistema.

Al encadenamiento regresivo se le suele llamar guiado por objetivos, ya que, el sistema comenzará por el objetivo (parte acción de las reglas), y operará retrocediendo para ver cómo se deduce ese objetivo partiendo de los datos. Esto se produce directamente o a través de conclusiones intermedias o subobjetivos. Lo que se intenta es probar una hipótesis a partir de los hechos contenidos en la base de datos y de los obtenidos en el proceso de inferencia.

En la mayoría de los sistemas expertos se utiliza el encadenamiento regresivo. Este enfoque tiene la ventaja de que el sistema va a considerar únicamente las reglas que interesan al problema en cuestión. El usuario comenzará declarando una expresión E, y el objetivo del sistema será establecer la verdad de esa expresión.

Para ello se pueden seguir los siguientes pasos:

1. Obtener las reglas relevantes, buscando la expresión E en la parte acción (éstas serán las que puedan establecer la verdad de E).

2. Si no se encuentran reglas para aplicar, entonces no se tienen datos suficientes para resolver el problema; se termina sin éxito, o se piden al usuario más datos.

3. Si hay reglas para aplicar, se elige una y se verifica su parte condición C con respecto a la base de datos.

4. Si C es verdadera en la base de datos, se establece la veracidad de la expresión E y se resuelve el problema.

5. Si C es falsa, se descarta la regla en curso y se selecciona otra regla.

6. Si C es desconocida en la base de datos (es decir, no es verdadera ni falsa), se le considera como subobjetivo y se vuelve al paso 1 (C será ahora la expresión E)

Existen también enfoques mixtos, en los que se combinan los métodos guiados por datos con los guiados por objetivos.

El éxito de un sistema experto depende casi exclusivamente de su base de conocimientos

CATEGORÍA	TIPO DE PROBLEMA	USO
Interpretación	Deducir situaciones a partir de datos observados.	Análisis de imágenes, reconocimiento del habla, inversiones financieras.
Predicción	Inferir posibles consecuencias a partir de una situación.	Predicción meteorológica, previsión del tráfico, evolución de la Bolsa.
Diagnóstico	Deducir fallos a partir de sus efectos.	Diagnóstico médico, detección de fallos en electrónica
Diseño	Configurar objetos bajo ciertas especificaciones.	Diseño de circuitos, automóviles, edificios, etc.
Planificación	Desarrollar planes para llegar a unas metas.	Programación de proyectos e inversiones. Planificación militar.
Monitorización o supervisión	Controlar situaciones donde hay planes vulnerables.	Control de centrales nucleares y factorías químicas.
Depuración	Prescribir remedios para funcionamientos erróneos.	Desarrollo de software y circuitos electrónicos.
Reparación	Efectuar lo necesario para hacer una corrección.	Reparar sistemas informáticos, automóviles, etc.
Instrucción	Diagnóstico, depuración y corrección de una conducta.	Corrección de errores, enseñanza.
Control	Mantener un sistema por un camino previamente trazado. Interpreta, predice y supervisa su conducta.	Estrategia militar, control de tráfico aéreo
Enseñanza	Recoger el conocimiento y mostrarlo	Aprendizaje de experiencia.

Tabla 2. Modelos funcionales de los sistemas expertos.

El interfaz de usuario permite que el usuario pueda describir el problema al sistema experto. Interpreta sus preguntas, los comandos y la información ofrecida. A la inversa, formula la información generada por el sistema, incluyendo respuestas a las preguntas, explicaciones y justificaciones. Es decir, posibilita que la respuesta proporcionada por el sistema sea inteligible para el interesado. También puede solicitar más información si le es necesaria al sistema experto. En algunos sistemas se utilizan técnicas de tratamiento del lenguaje natural, para mejorar la comunicación entre el usuario y el sistema experto.

La mayoría de los sistemas expertos contienen un módulo de explicación, diseñado para aclarar al usuario la línea de razonamiento seguida en el proceso de inferencia. Si el usuario pregunta al sistema, cómo ha alcanzado una conclusión, éste le presentará la secuencia completa de reglas usada. Esta posibilidad de explicación es especialmente valiosa, cuando se tiene la necesidad de tomar decisiones importantes amparándose en el consejo del sistema experto. Además, de esta forma, y con el tiempo suficiente, los usuarios pueden convertirse en especialistas en la materia, al asimilar el proceso de razonamiento seguido por el sistema. El subsistema de explicación también puede usarse para depurar el sistema experto durante su desarrollo.

El módulo de adquisición del conocimiento permite que se puedan añadir, eliminar o modificar elementos de conocimiento (en la mayoría de los

CORREO LECTORES

Para cualquier duda referente a los temas/artículos tratatos en la revista, escriba una carta a:

Sólo Programadores
Referencia: *Correo Lectores*
TOWER COMMUNICATIONS
C/ Marqués de Portugalete, 10 Bajo
Madrid 28027

COMO SUSCRIBIRSE A



Suscríbase enviando este cupón por correo o fax (91) 320.60.72, o llamando al teléfono (91) 741.26.62 Horario 9 a 14 y 15:30 a 18:30 h.

Deseo suscribirme a la revista SÓLO PROGRAMADORES acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año + regalo (filtro monitor) **por sólo** 10.995 ptas. Estudiantes carreras técnicas 40% Dto.: 8.250 ptas.

Nombre y apellidos..Domicilio..

Población..C.P ....Provincia......................Telf..............................Profesión....................

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº ....

Fecha de caducidad de la tarjeta................Nombre del titular, si es distinto..

☐ Domiciliación bancaria.

Señor Director del banco ....

Población ....

CODIGO CUENTA CLIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

Ruego a vd. que se sirva cargar en mi ☐ cuenta corriente ☐ libreta de ahorro número..

el recibo que le será presentado por TOWER COMMUNICATIONS, S.R.L. como pago de mi suscripción a la revista SÓLO PROGRAMADORES.

☐ Contra-reembolso del importe más gastos de envío.

☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.

☐ Giro Postal (adjunto fotocopia del resguardo).

Firma:

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Marques de Portugalete 10, Bajo
28027 Madrid.

casos reglas) en el sistema experto. Si el entorno es dinámico es muy necesario, puesto que, el sistema funcionará correctamente sólo si se mantiene actualizado su conocimiento. El módulo de adquisición permite efectuar ese mantenimiento, anotando en la base de conocimientos los cambios que se producen.

EJEMPLO DE FUNCIONAMIENTO DEL MOTOR DE INFERENCIA

Para ilustrar cómo trabajan los procedimientos de inferencia, se supondrá un sistema que contiene las siguientes reglas en la base de conocimiento:

R1: SI abrigo ENTONCES bingo

R2: SI chaqueta ENTONCES dentista

R3: SI bingo ENTONCES esposa

-Enfoque guiado por datos (o encadenamiento hacia adelante):

El problema es determinar si se da esposa sabiendo que se cumplen abrigo y chaqueta.

Lo primero que se hace es introducir en la base de datos abrigo y chaqueta:

B.D. = {abrigo, chaqueta}

El sistema identifica las reglas aplicables:

R = {R1, R2}

Selecciona R1 y la aplica. Esto genera bingo que se añade a la base de datos:

B.D. = {abrigo, chaqueta, bingo}

Como no se ha solucionado el problema, vuelve a identificar un conjunto de reglas aplicables (excepto la ya aplicada, que no cambiar'a el estado de la base de datos):

R = {R2, R3}

Selecciona y aplica R2 quedando:

B.D. = {abrigo, chaqueta, bingo, dentista}

El problema todavía no se ha solucionado, luego el sistema selecciona otro conjunto de reglas aplicables:

R = {R3}

Seleccionando y aplicando R3, la base de datos queda:

B.D. = {abrigo, chaqueta, bingo, dentista, esposa}

Como esposa se encuentra en ella, se ha llegado a la solución del problema.

-Enfoque guiado por objetivos (o encadenamiento hacia atrás):

Se quiere determinar si se cumple esposa teniendo en la base de datos abrigo y chaqueta:

B.D. = {abrigo, chaqueta}.

El conjunto de reglas aplicables en este caso será:

R = {R3} (R3 es la única que tiene esposa en su parte derecha).

Se selecciona R3 y se genera bingo. Como no se encuentra en la base de datos (no es verdadero ni falso) se le considera como subobjetivo.

El módulo de explicación muestra la secuencia de razonamiento seguida para resolver un problema

El sistema intentará ahora probar bingo.

Identifica las reglas aplicables:

R = {R1}.

Aplica R1 y se obtiene abrigo que es verdadero en la base de datos. De esta forma, se prueba el subobjetivo, y al probar éste, se prueba esposa resolviendo el problema.

UNA BASE DE CONOCIMIENTO EN C

Para concluir de manera práctica, se incluye un programa realizado en Turbo C 2.0 que permitirá la creación de una base de conocimientos basada en reglas. El lector puede ir añadiendo los conocimientos que considere. En el próximo número se entregará un motor de inferencia para poder razonar con las reglas introducidas.

El código fuente se encuentra en el fichero BASE1.C. El fichero BASE1.EXE es el programa ejecutable.

La estructura de la base de conocimientos consiste en un array de reglas:

```
struct regla baseco[TOTAL];
```

Cada regla está compuesta por una serie de condiciones (parte SI de la regla) y por una acción (parte ENTONCES de la regla):

```
struct regla
    {
    struct condicion *lcond; /* lista de condiciones */
    char accion [20]; /* acción */
    } re;
```

Las condiciones de una regla formarán una lista enlazada:

```
struct condicion
    {
    char cond [20];
    struct condicion *otrac; /* lista enlazada */
    } co;
```

El procedimiento basedereglas(), lee la parte condición y la parte acción de cada regla, guardando la información en la base de conocimientos. Sigue leyendo reglas o condiciones hasta que se introduzca una línea en blanco pulsando INTRO.

El programa tiene un funcionamiento muy sencillo. Como ejemplo, se va introducir la siguiente regla:

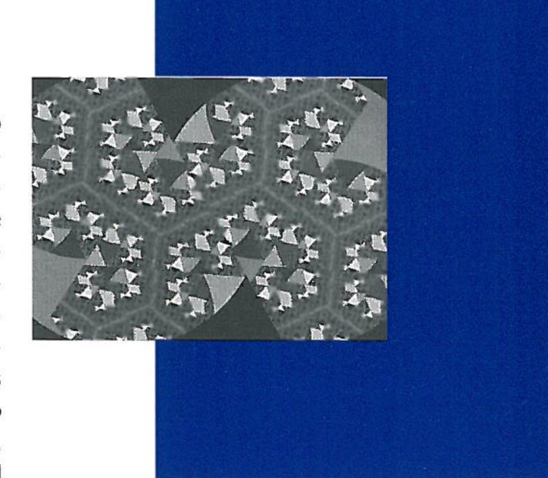
SI alargado Y amarillo ENTONCES plátano

Se selecciona la opción i en el menú. Se introduce “alargado” y a continuación “amarillo”. Como no hay más condiciones, se pulsa INTRO. Ahora se introduce “plátano”, que es la parte conclusión de la regla. El programa solicitará otra regla. Como en este ejemplo no hay más que una, se pulsa de nuevo INTRO para volver al menú.

El fichero FRUTAS es una pequeña base de conocimiento de ejemplo. Para cargarla se debe usar la opción c del menú. Las reglas de la base se podrán visualizar seleccionando v. ■

TÉCNICAS DE COMPRESIÓN FRACTAL

Francisco Otero



Con la llegada de la multimedia al PC, todo un mundo nuevo de posibilidades se ha abierto a los usuarios. Las tarjetas de vídeo de 24 bits son "moneda corriente" en los modelos actuales de PC, y los scanner e impresoras color de alta calidad tienen un precio asequible para una pequeña empresa dedicada a la multimedia. Sin embargo, un problema sigue sin ser solucionado; la falta de capacidad de los dispositivos actuales para almacenar a bajo precio toda esta gran cantidad de información. Por ejemplo, una imagen de 1024x768 pixels truecolor (a 16 millones de colores) sin comprimir ocupa 2.3 Mb de espacio en disco, y un minuto de animación varias decenas de Mb. Con este panorama, se ve la importancia que ha tomado el tema de la compresión de la imagen en los últimos años. Es necesario un método de almacenamiento, que nos permita un acceso rápido a la información y a la vez una relación de compresión muy elevada.

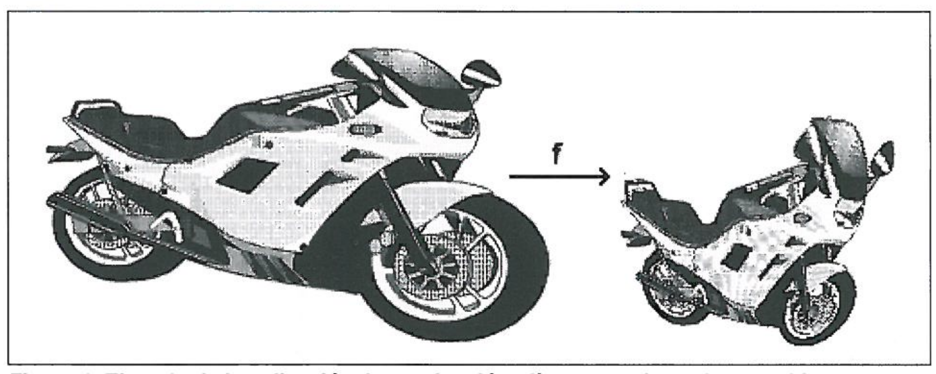


Figura 1. Ejemplo de la aplicación de una función afín contractiva sobre un objeto.

DISTINTOS MÉTODOS DE COMPRESIÓN

Las tecnologías de compresión de imágenes pueden dividirse en dos grupos: lossless y lossy. El primer método se caracteriza por que la imagen comprimida se puede descomprimir sin perdida ninguna de calidad, mientras que los métodos lossy alcanzan una relación de compresión mucho mayor, pero a costa de una ligera pérdida de calidad a la hora de descomprimir la imagen. Al primer grupo pertenecen los formatos gráficos tradicionales, como pueden ser el PCX, GIF, TIFF o el BMP, y el formato con pérdida de calidad más conocido es el JPEG.

Podría extrañar la utilidad de un método de compresión que no respete totalmente la calidad del original. Sin embargo esta desventaja se ve "suavizada" teniendo en cuenta que, mientras los métodos de compresión tradicionales raramente sobrepasan una relación de compresión de 3 a 1, el JPEG puede llegar a una relación de compresión de 25 a 1 con una calidad aceptable. A partir de ahí, la calidad de la imagen almacenada en formato JPEG sufre perdidas considerables.

Este articulo tratará de explicar una de las técnicas más recientes en el campo de la compresión de imágenes y que permiten obtener relaciones de

A pesar del relativo bajo precio de los sistemas de almacenamiento, los sistemas de compresión son uno de los campos a los que se dedican mayores esfuerzos.

compresión superiores al JPEG, con una pérdida de calidad casi siempre menor, se trata de la compresión de imágenes con técnicas fractales.

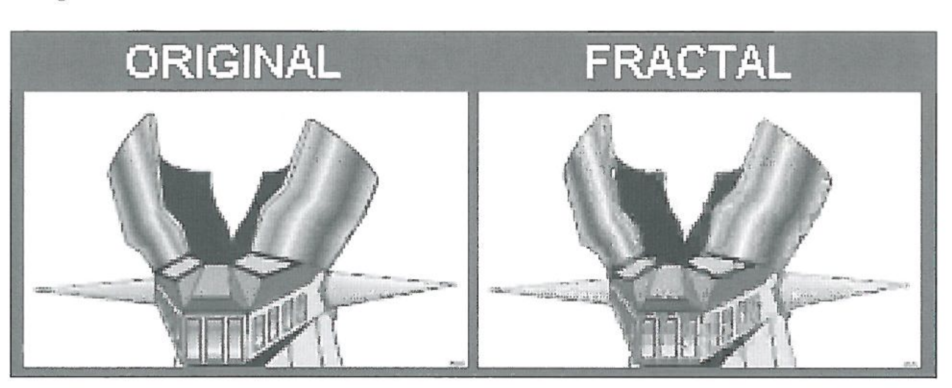


Figura 3. Compresión utilizando sólo 4 posibles transformaciones por dominio.

CAPACIDAD DE LOS FRACTALES PARA REPRESENTAR OBJETOS

Aunque la base matemática necesaria, para un estudio minucioso de las técnicas de compresión utilizando fractales es bastante extensa y compleja, y no es éste el tema que ahora nos ocupa, sí serán necesarias unas nociones básicas, antes de entrar de lleno en el estudio del algoritmo de compresión.

Debido a la capacidad de los fractales para la construcción de modelos muy complejos a través de procesos muy simples, son extraordinariamente útiles a la hora de modelizar, y explorar ciertos fenómenos de la naturaleza cuya complejidad viene determinada, por la repetición casi infinita de elementos muy simples. Si se observan las nubes, un bosque, montañas, o la textura de las rocas, se da uno cuenta, que están compuestos por estructuras muy semejantes repetidas casi indefinidamente a distintas escalas.

La palabra fractal fue acuñada por Benoit Mandelbrot como una estructura con formas semejantes, pero de distintos tamaños. Por ejemplo, un árbol está formado por ramas de gran tamaño, que a su vez están compuestas por otras más pequeñas y éstas, por otras aún más diminutas, así varios niveles. Intuitivamente se descubre que, con la información necesaria para representar el primer nivel de las ramas de un árbol, y aplicándolo reiteradamente, se puede representar el árbol completo con todo detalle, y posteriormente un bosque poblado de ellos.

El ejemplo anterior nos permite observar que los fractales resultan de gran utilidad a la hora de representar objetos en los cuales la información contenida es reducida, pero que ésta se aplica infinitas veces para formar un objeto de gran complejidad. Este fenómeno se conoce con el nombre de autosemejanza, y es una propiedad universalmente extendida en la naturaleza.

Con la llegada de las aplicaciones multimedia la compresión de imágenes busca nuevos horizontes

Considerando que los fractales resultan adecuados para representar objetos naturales, lo mismo que la línea y el círculo representan de alguna manera a los objetos creados por el hombre, es necesario un método que nos permita captar las situaciones del mundo real y transportarlas, mediante algún algoritmo, a una representación fractal adecuada. Con esto se consigue eliminar toda esa información redundante que poseen todas las escenas naturales, almacenar lo esencial, y posteriormente volver a restaurar la escena con toda su nitidez original mediante un proceso de descompresión adecuado.

APROXIMACIÓN DE IMÁGENES MEDIANTE FRACTALES

El método ideado por M.F. Barnsley, permite encontrar un fractal que se aproxime, tanto como se desee, a cualquier objeto natural (por muy irregular que éste sea). Permite obtener, a

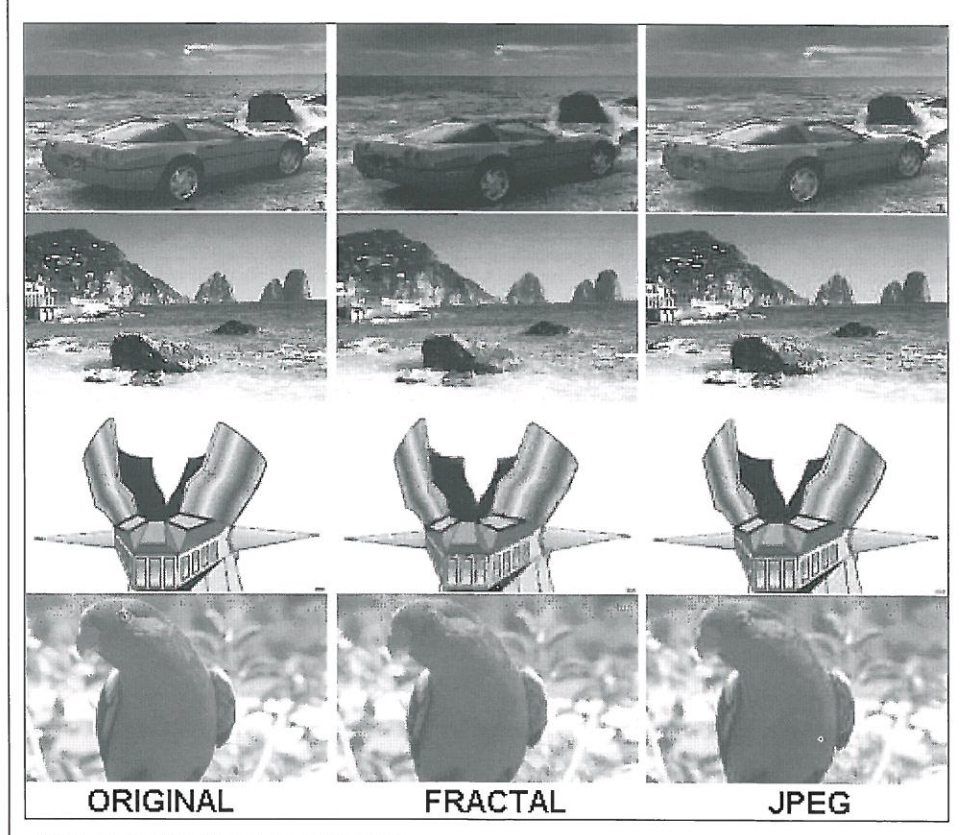


Figura 5. Comparación de resultados.

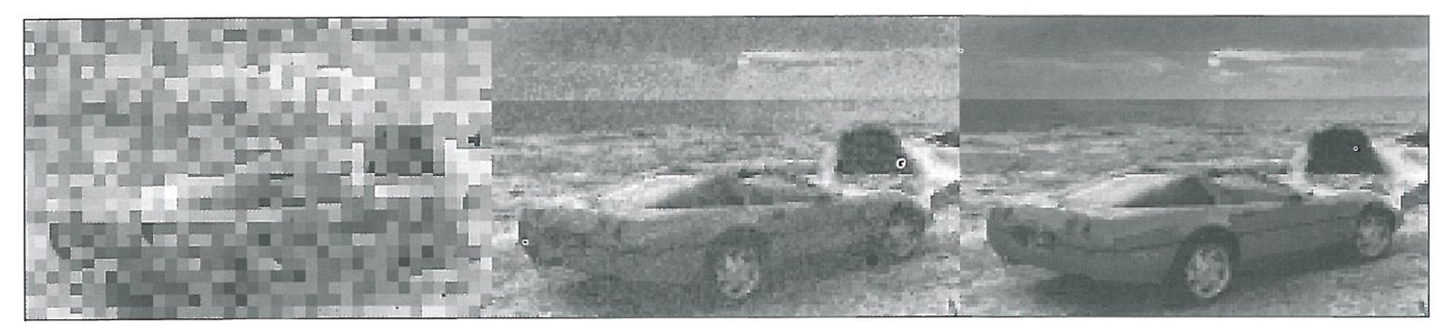


Figura 6. Proceso iterativo de decodificación de las imágenes fractales.

partir de una imagen natural, una familia de "contracciones" que generan un fractal, que se aproximará a la imagen natural tanto como se desee. Esta posibilidad de aproximar una imagen mediante conjuntos fractales, permite guardar esta familia de contracciones y posteriormente generarla en el momento adecuado. Esto supone un gran ahorro de espacio con respecto a los métodos clásicos.

Una aplicación contractiva, o contracción, es aquella que acerca los puntos, contrayendo las figuras, como muestra la figura 1.

Entre dos figuras del plano semejantes siempre existe una aplicación contractiva, que convierte la mayor en la menor. Esta aplicación contractiva es una composición de isometrías (translaciones, giros y simetrías), y una homotecia contractiva. Cuando una figura se obtiene de otra mediante giros, traslaciones, homotecias, simetrías y estiramientos, o comprensión en el sentido de algunos ejes, la transformación se denomina transformación contractiva afín.

Este concepto es fundamental es la compresión fractal. Una transformación contractiva W, puede definirse bidimensionalmente como:

$$W(x,y) = (ax + by + e, cx + dy + f)$$

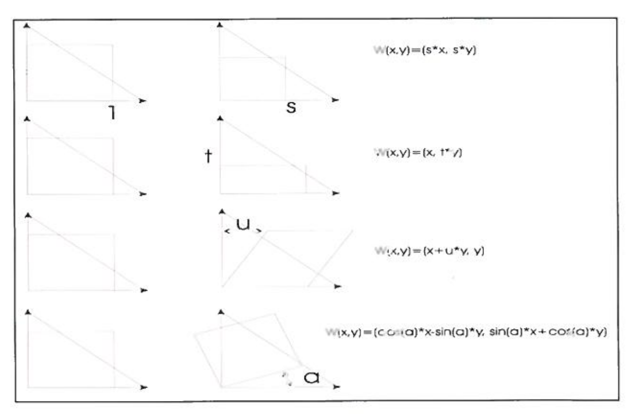


Figura 2. En este gráfico pueden observarse distintas transformaciones afines en 2 dimensiones.

Los coeficientes a, b, c y b determinan la rotación, deformación y escalado, siendo e y f los que determinan la traslación. Esta transformación mueve el punto (0,0) a (e, f), el punto (1,0) a (a + e, c + f), el (1,0) a (b + e, d + f) y el punto (1,1) a (a + b + e, c + d + f). Los valores a, b, c, d, e, f son los coeficientes afines para la transformación. En la figura 2 se pueden observar posibles transformaciones geométricas, así como su definición matemática.

Para la compresión se necesitan funciones contractivas tridimensionales, ya que una imagen se caracteriza por tener ancho, alto y un color por pixel. Aparte de realizar transformaciones geométricas, también resultará interesante aplicar cambios de brillo o contraste, para lo cual necesitaremos añadir otra dimensión más a las funciones contractivas que utilizaremos.

Ahora que se conocen los fundamentos matemáticos necesarios, ya se puede entrar de lleno en el diseño de un algoritmo para la compresión de imágenes reales. Como el lector seguramente intuirá, la piedra angular de la compresión serán las funciones contractivas afines, y la autosemejanza de las imágenes reales, que permitirá eliminar la redundancia intrínseca a los modelos naturales, y buscar la transformación contractiva adecuada que

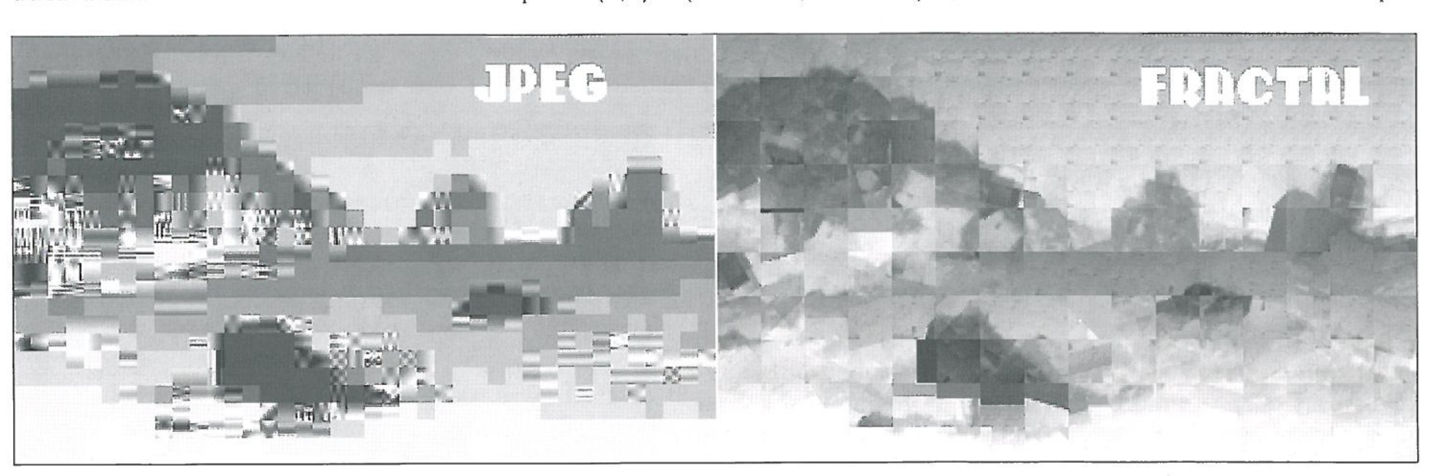


Figura8. Ultracompresión!. Una imagen JPEG 320x200 en 1627 bytes, formato fractal en sólo 640 bytes.

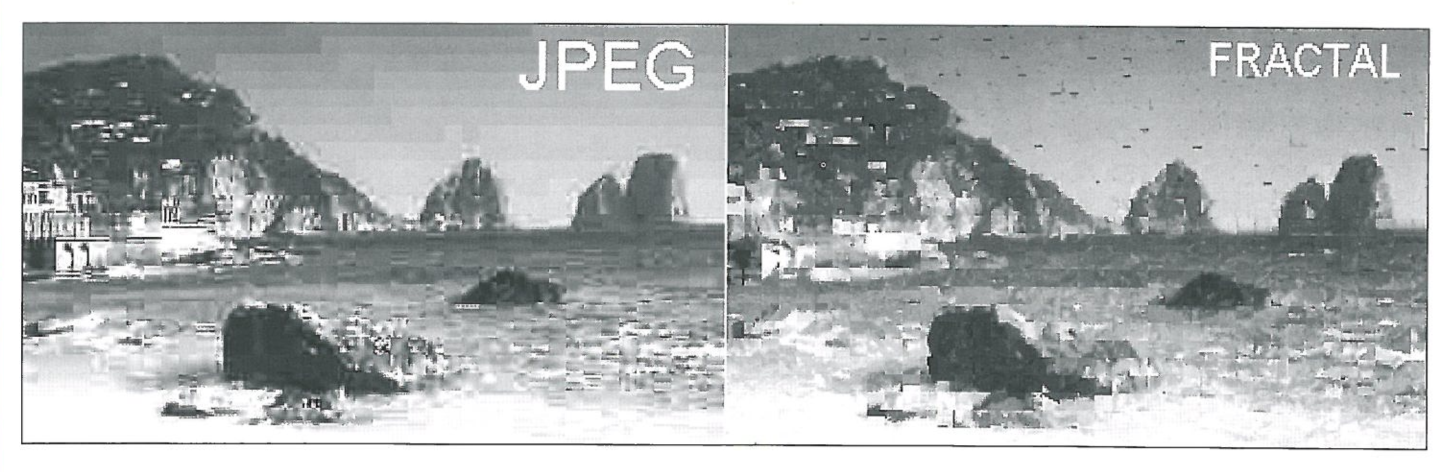


Figura7. JPEG 320x200 (3410 bytes) vs. FRACTAL 160x100 descomprimido a 320x200 (4000 bytes, pero comprimidos con PKZIP 3360 vs 3282).

permita representar cada región de la imagen como rotaciones, traslaciones, escalados, cambios de color o de contraste, etc, de otras regiones de la misma. Una vez encontradas esas transformaciones contractivas, que representan cada parte de la imagen en función de las restantes, se tiene la formula matemática que la representa. Lo mismo que conociendo la ecuación de una recta y=ax+b se puede representar a cualquier resolución, si se conocen los coeficientes de las transformaciones que representan la imagen, se puede representar una imagen fractal, que sea casi igual a la original, a cualquier resolución.

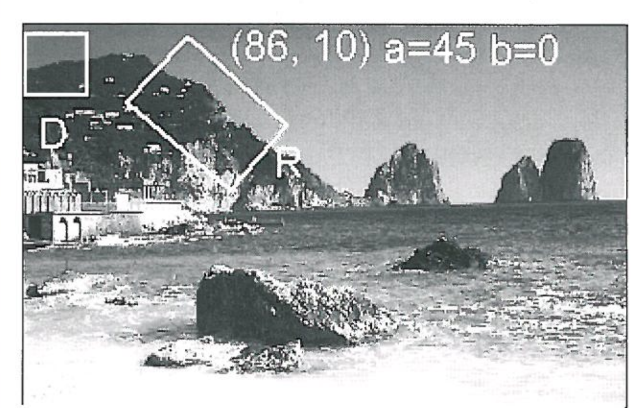


Figura 4. Explicación de cómo se obtienen los coeficientes de las transformaciones, y su significado.

DESCRIPCIÓN DEL ALGORITMO DE COMPRESIÓN

El primer paso en el proceso de compresión de una imagen consiste en dividirla en un conjunto de regiones que no se superpongan entre si y que cubran completamente a la imagen. Cada una de esas regiones, llamadas regiones del dominio, pueden ser de cualquier tamaño y de cualquier forma. Cada una de estas regiones se sustituirá por la trasformada afín que la genera. Lo primero a tener en cuenta, es que el tamaño de la imagen comprimida va a ser directamente proporcional al número de regiones del dominio, por lo que interesará, que cada una sea lo más grande posible, pero por otra parte, cuanto más grandes sean éstas, más difícil va a resultar obtener una función contractiva afín que, aplicada sobre otra parte de la imagen, se ajuste con precisión a la misma, por lo cual, con un numero pequeño de regiones del dominio obtenemos una gran compresión, pero una apreciable perdida de calidad. Es necesario encontrar un equilibrio entre el tamaño de las regiones del dominio y la calidad que necesitemos.

Otro concepto es el de regiones rango, que son regiones de la imagen de tamaño mayor que las regiones del dominio, pueden superponerse y no tienen que cubrir totalmente a la imagen. Es importante que las regiones rango sean dos o tres veces más grandes que los dominios, ya que se buscan transformaciones contractivas que transformen esas regiones rango en regiones del dominio. Para cada dominio, el núcleo de la compresión consiste en elegir la región rango adecuada para la cual, al aplicar una transformada afín apropiada, obtengamos como resultado la región que más se aproxime a esa región del dominio. En otras palabras, se aplican distintas transformaciones afines sobre las regiones rango, y si la región transformada se ajusta satisfactoriamente a la región del dominio, almacenamos la transformación. Para cada región de la imagen original se debe de buscar la transformación que más aproxime cualquier otra región de la imagen, de mayor tamaño, a la misma.

Una vez han sido obtenido los coeficientes de las transformaciones afines apropiadas, consistentes en rotaciones, escalado, cambios de contraste, etc., ya tenemos los datos necesarios para crear un fichero fractal. Estos coeficientes pueden ser comprimidos poste-



riormente utilizando un algoritmo de compresión del tipo Huffman o LZW para ahorrar más espacio. Para comprender la potencia de este método veamos un ejemplo. Sea una imagen de 256 pixeles de ancho por 256 de alto y con 256 tonos de gris, su tamaño descomprimido es 65536 bytes. Se considera el caso más sencillo, en el que las regiones del dominio son cuadrados de 8 x 8 pixeles, es decir 1024 regiones dominio, y las regiones rango serán regiones de 16 x 16 pixeles que se superponen, por lo que tendremos 241x241 = 58081 cuadrados. Sean 4 las distintas formas de mapear o ajustar las regiones rango con las regiones dominio, con lo que se tendrán 4x58081 = 232324 posibilidades, para cada una de las 1024 regiones del dominio. Las 4 posibles transformaciones son: girar 90º, girar 180º, girar 270 y girar 360º. Es decir, para cada una de las 1024 regiones o dominios en las que se ha dividido la imagen, de 8x8 pixeles cada una, aplicamos las 4 transformaciones anteriores sobre las 58081 posibles regiones de 16x16 pixeles de la imagen, que son todas las posibles, y la transformación que más se ajuste la almacenamos. Una vez se ha obtenido la transformación adecuada para cada dominio y necesitando 6 bytes para almacenar cada transformación, el tamaño del fichero resultante será de 6144 bytes, siendo la relación de compresión resultante superior a 10:1.

En la figura 3 se puede ver el resultado del ejemplo anterior aplicado sobre una imagen de 160x100 pixeles, y utilizando dominios de 4x4 pixeles. Para guardar cada una de las 1024 transformadas afines se necesitan 4 bytes: un byte para la X, otro para la Y, otro byte para el ángulo de rotación y para indicar la cantidad en la que hay que aumentar o disminuir en brillo. La imagen original ocupa unos 16K, mientras que la imagen fractal comprimida 4K. El tiempo de compresión necesario para obtener la imagen fue de 2 horas.

Se observa en la figura 4 como el dominio D es ajustado correctamente por el rango R girado 45 grados (a=45) en sentido horario, y sin aumentar su luminosidad (b=0). Así pues, ese dominio se representa en el fichero de coeficientes como 4 bytes: 86, 10, 45, 0.

Seudocódigo de compresión de imágenes fractales:

1 : Partir la imagen original en los distintos dominios.

2 : Elegir un conjunto de regiones rango.

3 : Elegir el tipo de las transformaciones afines que se aplicarán.

4 : Apuntar a un dominio.

5 : Comparar los datos del dominio con los datos obtenidos después de aplicar todas las transformaciones afines sobre todas las regiones rango.

6 : Guardar los coeficientes de la transformación que minimiza el error.

7 : Si existen más dominios ir al paso 4.

8 : Comprimir el fichero resultante mediante un algoritmo de compresión estándar.

El método para seleccionar los dominios en el programa en C incluido de ejemplo es sencillo, ya que son cuadrados de 4 por 4 pixels. Una mejora sustancial sería utilizar un algoritmo adaptativo para la selección de las regiones del dominio, como el quadtree o el HV-partitioning, que utilizan dominios de tamaño variable dependiendo de la complejidad de la imagen.

Otro aspecto de obligada optimización es la velocidad de compresión. El tiempo de compresión de una imagen 160x100 pixels es de unas 3 horas en un 486 DX50.

Se puede aumentar sensiblemente la velocidad de compresión utilizando algún criterio para clasificar los dominios y las regiones rango, para evitar tener que procesar todos los rangos para cada dominio posible. Se pueden clasificar según características de contraste, rugosidad, bordes, etc. Un método que da buenos resultados es clasificarlos según su varianza estadística.

El algoritmo propuesto está preparado para comprimir imágenes en escala de grises. El método de compresión de color es el mismo, salvo que para comprimir imágenes en color es necesario comprimir cada uno de los tres colores primarios (rojo, verde, azul) por separado. Luego se combinan a la hora de la decodificación para restaurar los colores deseados. Como el ojo humano no es particularmente sensible a la información del color, se obtienen mejores resultados convirtiendo cada tripleta RGB (Red, Green, Blue) a su correspondiente YIQ(Luminancia, Tono, Saturación) mediante las ecuaciones:

$$Y = 0.299*R + 0.587*G + 0.114*B$$
$$I = 0.596*R - 0.247*G - 0.322*B$$
$$Q = 0.211*R - 0.523*G + 0.312*B$$

Los canales I y Q se denominan crominancia, y es posible comprimirlos más que el canal Y, sin una aparente degradación de la imagen. Por eso es posible obtener un relación de compresión más elevada en las imágenes en color que en las imágenes de tonos de grises, aunque el tiempo de compresión es unas tres veces más elevado.



Las ecuaciones para convertir el modelo YIQ a RGB son:

R = Y + 0.956*I + 0.621*Q
G = Y - 0.273*I - 0.647*Q
B = Y - 1.104*I + 1.701*Q

DESCOMPOSICIÓN DE IMÁGENES

La principal característica del método de compresión fractal, es que permite recrear una imagen comprimida, partiendo de un conjunto de coeficientes. Estos coeficientes indican, que cada parte de la imagen guarda una determinada relación, de forma o color, con otra región de la misma. Este método parecería bastante intuitivo, si no fuera por el hecho de que en el fichero comprimido no se guarda ninguna región de la imagen, sólo guardamos información de cómo unas regiones y otras se relacionan. La explicación a esta paradoja está en ciertos teoremas de la teoría de fractales relacionadas con los attractors, que no vienen al caso. Baste con decir, que un attractor es la figura a la que converge un punto del plano al que aplicamos, repetidas veces y en un orden aleatorio, un conjunto finito de transformaciones geométricas.

La compresión fractal permite relaciones de compresión superiores al JPEG, y como mínimo, con la misma calidad

Michael Barnsley es uno de los pioneros en ésta avanzada técnica.

Se había comentado que la compresión fractal de una imagen es como obtener la ecuación que la representa. Al contrario que la compresión, la descompresión resulta muy rápida y posee la ventaja de poder decodificar la imagen original a cualquier resolución. El proceso de decodificación de los coeficientes obtenidos para obtener la imagen es el siguiente.

El primer paso es crear 2 buffers A y B en memoria, cada uno capaz de almacenar la imagen que se va a decodificar. El contenido inicial de los buffers no importa, pueden estar inicializados a cualquier valor. Luego se leen todos los coeficientes almacenados en el fichero, y se aplica cada transformación contractiva contenida en el fichero tomando como origen el buffer A y almacenamos el resultado de la transformación en el dominio correspondiente de B, así para todos los dominios. Una vez se ha realizado esta primera iteración, la imagen obtenida en B se compara con la que tenemos en A, si son muy distintas, se copia el contenido de B en A y se vuelve a repetir el proceso, aplicando secuencialmente las transformaciones sobre A, y almacenando el resultado en el buffer B. Se repite el proceso hasta que los buffers A y B sean idénticos, y esa será la imagen decodificada. En general, son necesarias unas 20 iteraciones para obtener la imagen final.

Es posible decodificar la imagen con una resolución distinta de la original, por ejemplo, si se quiere decodificar la imagen al doble del tamaño original, sólo se deben de reservar los buffers correspondientes del mayor tamaño y multiplicar todos los coeficientes de las transformaciones por 2. El proceso es igual al anterior.

Seudocódigo de decodificación de imágenes fractales:

1 : Crear dos buffers, uno para almacenar los dominios y otro para los rangos.

2 : Inicializar el buffer de rangos con un valor arbitrario.

3 : Elegir un dominio en el buffer de dominios.

4 : Aplicar la correspondiente transformación sobre el buffer de rangos y almacenar el resultado en el correspondiente dominio.

5 : Mientras halla dominios ir al paso 3.

6 : Una vez se han procesado todos los dominios, comparamos los dos buffers, si son iguales el proceso ha acabado, en otro caso volcar el buffer de dominios sobre el buffer de rangos y volver al paso 3.

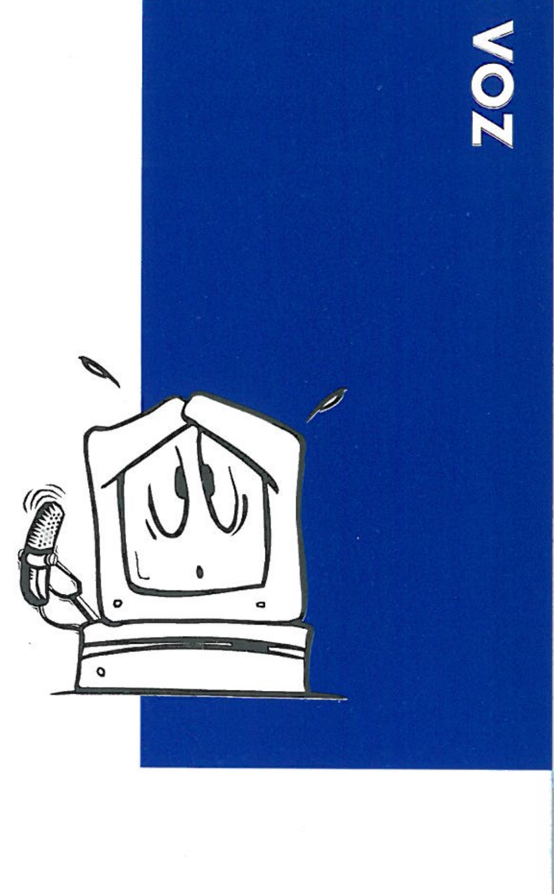
CONCLUSIÓN

Su principal característica es una elevada relación de compresión, unido a la rapidez y sencillez del proceso decodificación. Por lo tanto, es el formato perfecto para el almacenamiento, tanto de imágenes estáticas como de vídeo.

Su único inconveniente es la lentitud y complejidad del algoritmo de compresión, con lo que tardará algunos años en convertirse en un estándar para almacenamiento de imágenes; pero supone un gran avance respecto a los formatos tradicionales. Actualmente las únicas implementaciones comerciales se encuentran en hardware, lo cual limita su uso a los usuarios particulares. ■

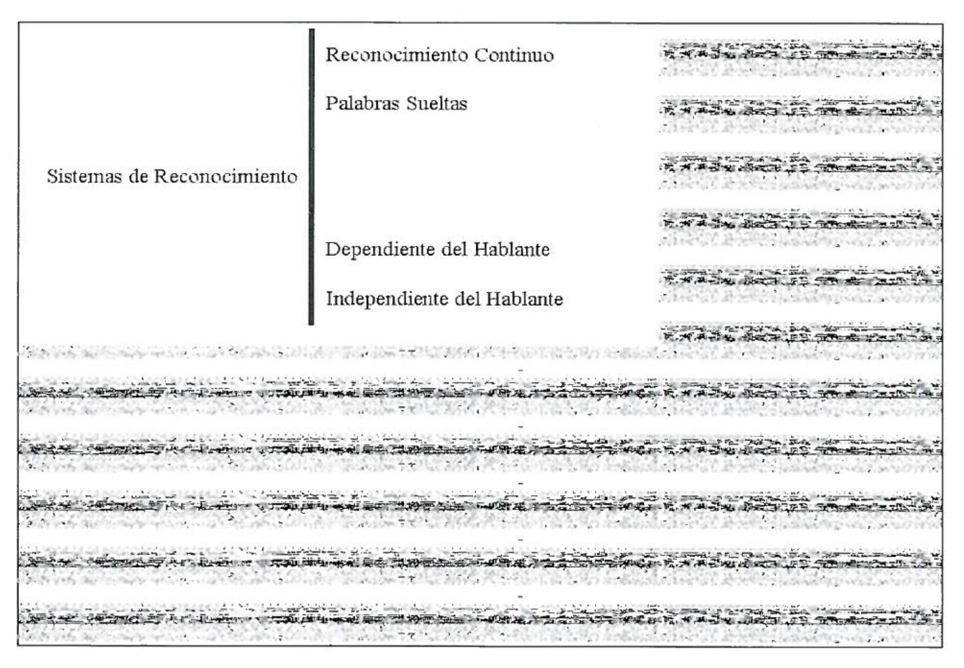
HALL ¿ME ESCUCHAS?

Julio Mateo



El reconocimiento de voz siempre se ha tenido como un tema tabú, reservado únicamente a trabajos de laboratorio, y con fama de ser algo muy dificultoso y fuera del alcance del programador medio. Con este artículo pretendemos iniciar el lector en este apasionante tema, y clarificar algunos conceptos. Veremos como con un simple PC y una tarjeta de sonido, podemos hacer que nuestro querido amigo tenga "orejas", y haga cierto el dicho: "responde a la voz de tu amo".

po de la inteligencia artificial. Hoy en día no es muy problemático que un ordenador reconozca la frase: "El día está soleado". El problema es que "entienda" lo que esa frase significa. Así que, para nosotros, el RV (Reconocimiento de Voz) será solamente el primer proceso, la acción por la cual un ordenador es capaz de reconocer la información hablada que el usuario suministra, para un posterior procesamiento. Como se muestra en la figura 1 tenemos diferentes tipos de RV. Existen sistemas



Pero, ¿qué se entiende por reconocimiento de voz?. Es un término difícil de definir, ya que el reconocer una voz conlleva dos acciones, el reconocimiento propiamente dicho de distinguir una palabra de las demás, y un segundo proceso de asimilación de la información obtenida. Es decir, que con esta segunda acción entramos en el cam-

capaces de reconocer palabras aisladas, y otros que separan las palabras de una frase para reconocerlas individualmente. Además algunos sistemas son capaces de entender palabras pronunciadas por cualquier persona (normalmente con un vocabulario muy limitado), y otros que tras un entrenamiento previo quedan personalizados

En este artículo se presentarán las bases del reconocimiento de la voz por medio de la combinación de una tarjeta de sonido y un ordenador. Emplearemos para ello las populares SoundBlaster.

PROBLEMAS:	
RUIDO:	Señal aditiva de ruido ambiental.
DISTORSIÓN:	Reverberaciones que alteran el espectro de la señal.
EFECTOS DE ARTICULACIÓN:	Pronunciaciones diferentes
CONDICIONES ADVERSAS DE CAPTACIÓN DE LA SEÑAL	

para un usuario determinado. En este artículo presentamos dos sistemas de RV (dependiente e independiente), para el reconocimiento de palabras sueltas.

HISTORIA DEL RV

Las primeras investigaciones sobre sistemas de RV por medio de máquinas se remontan a los años 50. En 1952, en los laboratorios Bell se desarrolló un sistema que reconocía dígitos aislados, pronunciados por una sola persona. En 1959, en el MIT se dispuso de un sistema reconocedor independiente de quién hablase.

Llegados a 1960, laboratorios japoneses entraron en acción desarrollando sistemas hardware, que permitieron un avance significativo en este campo, al aumentar la velocidad de los métodos empleados para este tipo de trabajos. También en estos años se desarrollaron sistemas de reconocimiento continuo de la voz, donde una persona podía hablar casi a velocidad normal, y el software empleado diferenciaba unas palabras de otras. Sin embargo, hasta los años 80, el reconocimiento continuo de la voz no fue un hecho palpable.

Los sistemas de amplio vocabulario y reconocimiento continuo fueron impulsados por DARPA, (Defense Advanced Research Projects Agency), a través de múltiples programas de investigación. De este proyecto surgieron algunos de los sistemas más avanzados que existen hoy en día, como el sistema SPHINX, considerado el mejor sistema de RV hasta la fecha.

PROGRAMAS DISPONIBLES PARA SISTEMAS PC

De los sistemas de RV de bajo coste que existen actualmente para el mercado de los ordenadores personales podemos destacar el programa Voice Assist, de la empresa Creative Labs., que entrega con sus tarjetas de sonido SoundBlaster. Es un sistema que sólo reconoce palabras aisladas, y es dependiente del interlocutor, aunque permite guardar varias configuraciones, para que no tener que entrenarse de nuevo al ser usado por una persona distinta. Otros programas que podemos usar en nuestras SB, sin requerir ningún hardware especial son el Voice Blaster e InVoice, de similares características al programa de Creative.

Como sistema más profesional tenemos el sistema de dictado personal de IBM. Es dependiente del interlocutor, pero es capaz de reconocimiento continuo, llegando a alcanzar las 100 palabras por minuto y cerca del 98% de precisión en el reconocimiento. Este sistema requiere de una tarjeta especial lo que encarece el producto, pero es una buena señal para los usuarios porque es el primer intento serio de hacer llegar el RV al gran público.

El propósito del RV es proporcionar un beneficio real de productividad al usuario

EL COMIENZO

El proceso por el cual emitimos un sonido comienza en el cerebro. Una vez que hemos pensado en la palabra, idea o acción que queremos transmitir, el cerebro envía órdenes a la boca, labios, lengua, etc. para que adopten la postura concreta que permitirá pronunciar la palabra deseada. En el reconocimiento, los sonidos son diferenciados por el cerebro a través de la información recibida a través del sistema auditivo; proceso que tenemos que imitar de forma artificial, y donde confluyen multitud de disciplinas que hay que conocer muy bien para llegar a buen término. Estas son:

- Procesamiento de señales: hay que extraer cierta información de las señales captadas, con lo que debemos procesarlas, pasarlas al domino de la frecuencia, filtrarlas, etc.
- Física (Acústica): hay que conocer los mecanismos por los que se produce la voz humana y la forma en la que los oídos la perciben y la diferencian.
- Reconocimiento de patrones: conjunto de algoritmos que nos permitirán



clasificar las muestras recibidas en patrones y compararlos entre si.

- Teoría de la información y la comunicación: métodos estadísticos que nos permitirán extraer información de los patrones recibidos y clasificarlos.

- Lingüística: estudiará las relaciones entre la fonética, sintaxis y semántica.

- Fisiología: estudia los procesos ocurridos en el sistema nervioso central y será de gran utilidad en la aplicación de las redes neuronales al RV

- Psicología: estudia las bases de la comunicación humana.

Para la aplicación a la que pensamos destinar nuestro programa no tendremos que ser expertos en estas materias, ya que veremos un método muy sencillo para obtener resultados razonables. Sin embargo, no nos libraremos de los problemas inherentes a todo reconocimiento como son los mostrados en la figura 2.

EL RECONOCIMIENTO DE VOZ

Tenemos tres aproximaciones principales al RV.

1. La aproximación acústica-fonética.
2. La aproximación por reconocimiento de patrones.
3. La aproximación por inteligencia artificial.

La aproximación acústica-fonética se basa, en la teoría que postula la existencia de unidades fonéticas finitas y distintas entre si en el lenguaje hablado, y que estas unidades fonéticas se caracterizan por un conjunto de propiedades que se manifiestan en la señal hablada, o en su espectro a lo largo del tiempo. En las figuras 3,4,5 y 6, podemos ver pronunciada la palabra "hola" por un hombre y una mujer en los dominios del tiempo y la frecuencia.

En la aproximación por reconocimiento de patrones, se utilizan patrones de voz sin ningún tipo de característica especial. Usualmente posee dos fases, una primera fase de entrenamiento donde se clasifica el sonido captado en patrones y una segunda, el reconocimiento propiamente dicho donde se comparan los nuevos patrones recibidos, para así poder determinar la voz. En una variante de esta técnica se basa el programa presentado en este artículo.

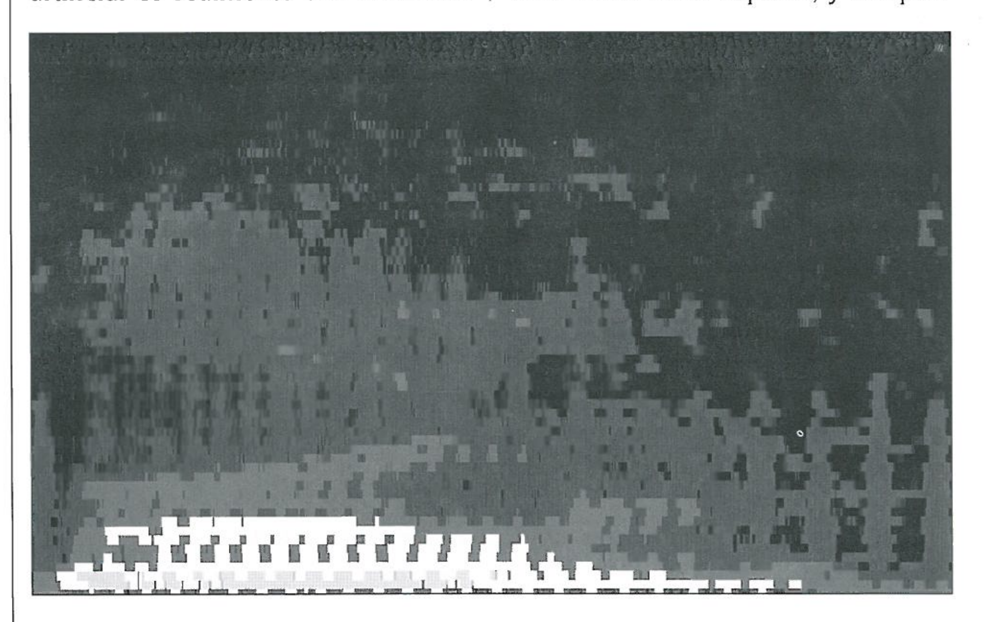
Este método es el más usado actualmente por las siguientes razones:

Las primeras investigaciones sobre RV por medio de máquinas se remontan a los años 50

* Simplicidad de uso.

* Es robusto y no varía frente a diversas circunstancias. Previo entrenamiento puede reconocer cualquier sonido, palabra o frase.

La aproximación por inteligencia artificial es realmente una combinación de los anteriores métodos, donde se usan las técnicas que se cree sigue el cerebro en el RV. En esta aproximación se usan ampliamente las redes neuronales.



LOS PROGRAMAS

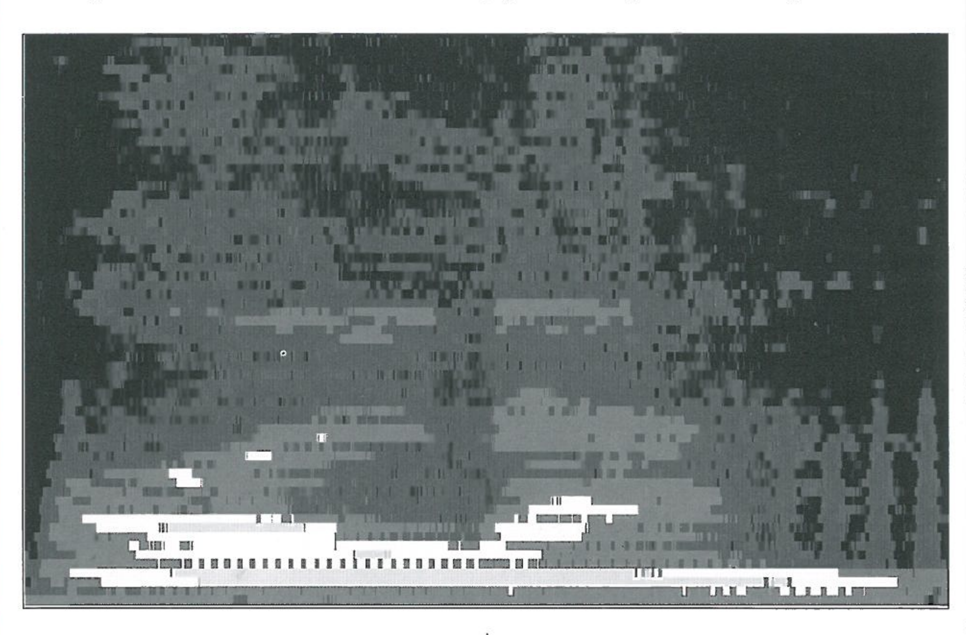
Presentamos dos programas donde se aplica lo expuesto anteriormente. Uno de ellos es el SBRECOG de Johannes Kiehl que aparece en varios CD y BBS. Este programa está escrito en C y es de dominio público. Utiliza una aproximación por reconocimiento de patrones. Es un programa muy potente, que con el entrenamiento adecuado puede alcanzar un índice de aciertos bastante elevado.

El programa comienza con una fase de entrenamiento donde pronunciamos varias palabras que el programa cuadratiza, esto es, convierte la señal analógica que se recibe en una señal cuadrática, con sólo dos valores posibles (véase figura 7). Luego prosigue con una clasificación de la señal en diferentes patrones, según el número de ceros que posee (las veces que la señal cruza por cero). En la fase de reconocimiento realiza el mismo proceso con la señal captada, y compara el patrón obtenido con los guardados anteriormente. En el proceso de comparación está la clave del reconocimiento. Este programa utiliza técnicas de clasificación y comparación de patrones muy avanzadas, que se salen del ámbito de este artículo. No obstante no es muy difícil de aplicarlo en nuestros programas con resultados efectivos.

El otro programa es una aproximación al reconocimiento de la voz con independencia del interlocutor. Tenemos por tanto un vocabulario muy reducido (tan sólo dos palabras) y aunque su índice de precisión es bastante alto no es muy difícil engañarlo, ya que prácticamente diferencia

una palabra larga de una corta.

En una primera fase del reconocimiento se procede a grabar en memoria las muestras resultantes de la digitalización del sonido. Estas muestras se realizan a una frecuencia aproximada de 11 Khz, más que suficiente para distinguir la voz humana. Pensemos que en telefonía se emplean sólo 8 Khz. Estas muestras tendrán una longitud máxima de 32000 bytes. La captura del sonido se realiza mediante una SoundBlaster. Tendremos así una tabla llamada muestra, que contiene 32000 valores resultantes de la digitalización del sonido.



A continuación procedemos a cuadratizar la señal, es decir, a partir de un valor determinado que hemos establecido en 128, convertimos la señal en 255, y si es menor, en 0. De esta forma, por cada muestra obtenemos una ristra de ceros y unos. Mediante este proceso, convertimos una señal que puede tener 255 niveles posibles a sólo dos, como veíamos en la figura 7.

Seguimos dividiendo la señal en 16 intervalos de 2000 muestras cada uno, y contamos el número de ceros que existe en cada intervalo. Este paso no es necesario, ya que lo que nos interesa es saber el número de ceros que posee toda la señal, pero acelera considerablemente los cálculos. Por último contamos el número de ceros que posee la señal.

Ya sólo queda la fase de reconocimiento. El programa distingue entre los números siete y dos. Estadísticamente pronunciando el número siete obtenemos 7.297 ceros, y pronunciando dos 3.697. Así que el programa calcula cuál de estos números es el que más se aproxima al que hemos pronunciado.

Este reconocimiento, no es tan preciso como el programa de Johannes, pero es un método rápido de control para distinguir entre dos palabras diferentes. Pensemos en un videojuego, donde guiemos una nave que puede moverse en dos direcciones (derecha e izquierda) y podamos controlarla por medio de la voz.

PROBLEMAS DE LA TÉCNICA EMPLEADA

* El ruido ambiente o de fondo puede confundir al programa. Puesto que es un ruido que se suma a la señal original, el número de ceros de la muestra puede alterarse significativamente.

* Al no realizarse un estudio en el dominio de la frecuencia de la señal, y al estar comparándolas según estos patrones, palabras muy parecidas pueden confundir al reconocedor: decir "seis" y "veis" serán reconocidas como las mismas palabras.

* Es muy sensible a la pronunciación, basta con pronunciar la palabra un poco diferente, de forma más rápida o pausada, en tono más agudo o grave para confundir al reconocedor.

HACIA EL FUTURO

Si se decide a desarrollar un sistema de reconocimiento de voz debe tener en cuenta las siguientes recomendaciones:

* El propósito del RV es proporcionar un beneficio real de productividad al usuario, ya que es la forma de comunicación habitual entre los humanos.

* Un sistema de RV debe ser amigable. Si con el RV tratamos de facilitar la vida al usuario no podemos dificultar el uso de este sistema.

* Debe ser eficaz, un sistema de RV que confunda palabras, o con una tasa de errores alta pondrá al usuario en su contra.

* Este sistema debe responder en tiempo real. Se dice que un sistema de RV responde en tiempo real, cuando el reconocimiento de la voz se obtiene 250 ms después de la voz hablada. Esto posibilita un diálogo continuo con la máquina.

Actualmente las aplicaciones más importantes del RV están en:

1. Oficinas. Aplicaciones de entrada de datos en formularios, control de bases de datos, etc.

2. Fabricación. Los trabajadores pueden controlar la producción por medio de la voz sin necesidad de estar pendientes en todo momento de las máquinas.

3. Telefonía y telecomunicaciones. Es actualmente el principal campo de aplicaciones de los sistemas de RV, y uno de los que más futuro poseen.

4. Medicina. Creación y edición de informes médicos.

5. Videojuegos. Permiten mayor interactividad con el usuario.

Esperamos que con este artículo haya quedado prendado con este fascinante campo del RV. Sólo hemos visto la punta del iceberg, y los resultados a los que podemos llegar con cualquier ordenador personal pueden llegar a ser espectaculares. ■

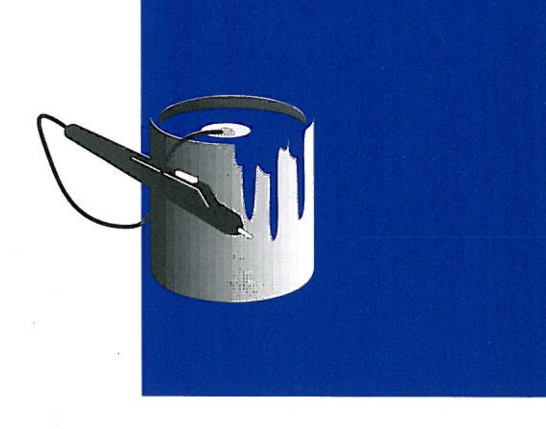
TRATAMIENTO DIGITAL DE IMÁGENES

Enrique Coiras

Existe un gran número de funciones de tratamiento de imágenes que pueden implementarse con un sencillo programa de ordenador. Aquí se verán algunas de las más útiles y fáciles de realizar: métodos de suavizado, afilado, eliminación de ruido y mejora de contraste. Aunque todas estas funciones estarán orientadas a imágenes en blanco y negro, los métodos descritos son fácilmente extensibles a imágenes en color.

En cualquier caso, el que sólo se puedan representar 64 niveles de gris no impide trabajar con imágenes de 256 niveles o más. De hecho, durante el desarrollo del artículo se verá que en ocasiones hay que convertir las imágenes a valores reales para poder efectuar determinados cálculos.

Para poder manejar imágenes de distinto tipo y rango de valores, es necesario introducir el concepto de normalización.



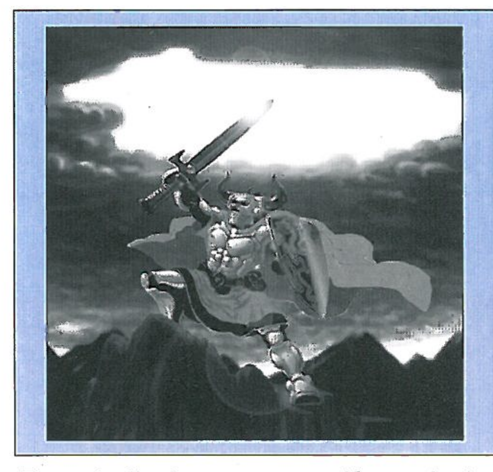




Figura 1 : Una imagen y su versión suavizada con desenfoque gaussiano de 5 pixels

NIVELES DE GRIS

Los puntos de una imagen digital en blanco y negro sólo pueden tomar valores (llamados niveles de gris) dentro de un rango determinado, dependiendo del número de bits por pixel que tenga la imagen. Para imágenes de 8 bits por pixel, que son las que se usarán en este artículo, el número de niveles de gris posibles es de 256. También hay que considerar las limitaciones de nuestro hardware : una VGA típica sólo puede representar 64 niveles de gris, en baja resolución, o 16 en alta.

NORMALIZACIÓN

Normalizar una imagen es modificar su rango de valores para que se encuentre entre unos límites máximo y mínimo determinados. Por ejemplo, cuando se trabaja con imágenes de 8 bits por pixel, y una función nos devuelve una imagen en valores reales comprendidos entre, digamos, 3.5 y 6.8, debemos convertir ese rango al de (0, 255) para poder aprovechar al máximo las posibilidades de nuestro formato de almacenamiento.

La potencia de los ordenadores domésticos permite realizar funciones de tratamiento de imágenes, que hasta hace poco estaban reservadas exclusivamente a las estaciones de trabajo.

La normalización no es más que una transformación lineal del nivel de gris de cada punto de la imagen :

ng' =ng*((max' - min') / (max - min))+ min'

donde :

ng es el nivel de gris original,
ng' es el nuevo nivel de gris,
min y *max* son los límites del rango de valores original, y
min' y *max'* son los límites del rango de valores final

Para normalizar un imagen por tanto, habrá que hallar primero sus valores máximo y mínimo. Después bastará con aplicar la fórmula anterior a cada punto, especificando los valores límite deseados.

VENTANAS Y MÁSCARAS

Un gran número de funciones de tratamiento digital se realizan considerando individualmente pequeñas porciones de imagen, o ventanas alrededor de cada punto. Frecuentemente, cada uno de los puntos de la ventana, debe ser multiplicado por un cierto número que le dará un peso en la operación que se esté realizando. A la matriz de pesos que modifican una ventana se le suele llamar máscara y al proceso de multiplicar la máscara por ventanas alrededor de cada punto de la imagen se le llama convolución. Las funciones que se pueden implementar utilizando máscaras y ventanas son muy diversas : suavizado, eliminación de ruido, extracción de bordes, adelgazamiento de contornos, etc.

Algo que debemos tener en cuenta al utilizar funciones de este tipo, es que siempre existirá una zona, en los bordes de la imagen, en la que la operación no podrá ser aplicada correctamente, ya que para los pixels de los extremos no podemos coger una ventana que se encuentre contenida en la propia imagen. Existen varias técnicas para superar esta dificultad :

1) Hacer una imagen más grande que contenga a la nuestra y aplicar la operación sobre ella, recortando luego nuestra imagen modificada o, lo que es lo mismo, rellenar las partes de las ventanas que no podamos coger con ceros (u otro valor cualquiera) y hacer el cálculo normalmente. Estos métodos darán valores poco útiles en los bordes de la imagen resultado, pero para cierto tipo de aplicaciones, como las artísticas, puede que eso tampoco sea demasiado preocupante. Otra variación más refinada de este método consiste en duplicar los puntos de los extremos para rellenar la nueva zona de bordes que hemos incluido.

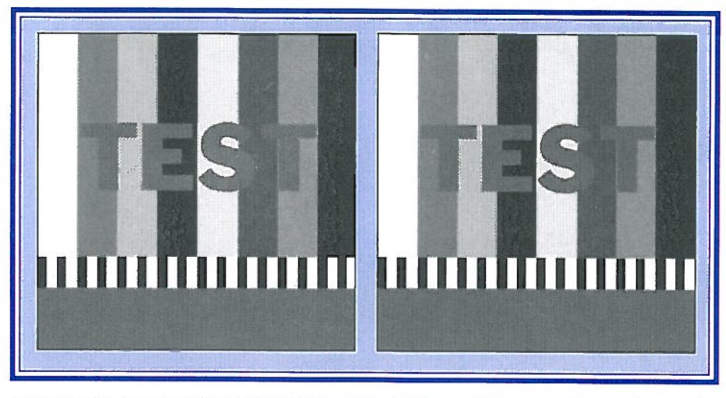
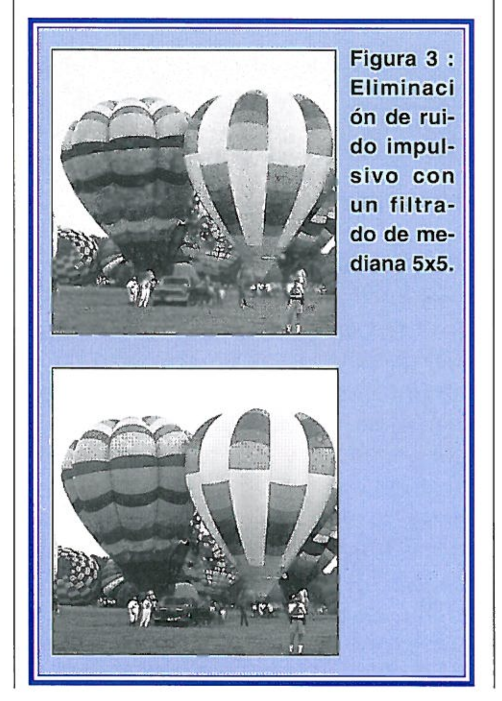


Figura 2 : Un patrón de prueba con ruido uniforme antes y después de pasarle un filtro de media restringida.

2) Considerar que la imagen es topológicamente equivalente a una superficie toroidal, es decir, suponer que saliendo por un lado de la imagen se entra por el contrario (por ejemplo, el siguiente punto a la derecha de un punto del borde derecho de la imagen, sería el punto del borde izquierdo que esté a la misma altura que el primero). Esto suele ser útil si estamos interesados en que las imágenes tratadas puedan adosarse periódicamente (para decorar el escritorio de Windows, por ejemplo).

3) No aplicar la operación en las zonas en las que no pueda aplicarse, que, además de ser la solución más sencilla, es la única aceptable cuando se están desarrollando aplicaciones técnicas.


Figura 3 : Eliminación de ruido impulsivo con un filtrado de mediana 5x5.

Las funciones que acompañan al artículo utilizan este último método.

MÉTODOS DE SUAVIZADO

Una de las operaciones más fáciles de programar es el suavizado o emborronado de una imagen. La implementación más sencilla consiste, en sustituir el valor de cada punto por la media de los valores de una ventana de tamaño nxn centrada en él. Cuanto más grande sea n, mayor será el emborronamiento conseguido (aunque en la práctica, para emborronar más, lo que se hace es pasar varias veces el suavizado de media 3x3).

Puede verse fácilmente que el suavizado de media es equivalente a la convolución con la máscara {{1/n^2, ..., 1/n^2}, ..., {1/n^2, ..., 1/n^2}}, así que podemos utilizar la función de convolución explicada anteriormente para realizar el suavizado. Claro que, de este modo el procedimiento será más lento que si se calcula directamente la media de la ventana.

Una forma más correcta de suavizar una imagen es hacer la convolución con una gaussiana, que valora la contribución de cada punto de la ventana en función de su distancia al centro de la misma. Podría decirse que es un suavizado "mejor hecho", aunque a simple vista no se pueda distinguir cual de los dos métodos se ha usado para emborronar una imagen determinada. Sin embargo, un caso en el que se ve fácilmente por que es mejor la gaussiana, es en el de un punto blanco sobre un



fondo negro : al suavizar por el método de la media se obtendría como resultado un cuadrado de intensidad constante, mientras que usando el desenfoque gaussiano saldría una mancha circular de intensidad decreciente según nos alejamos del punto (que es precisamente lo que se espera del emborronamiento de un punto).

La fórmula de la gaussiana es la siguiente :

gauss (x, y) = k * exp (- kx * (x - x0)^2) * exp (- ky * (y - y0)^2)

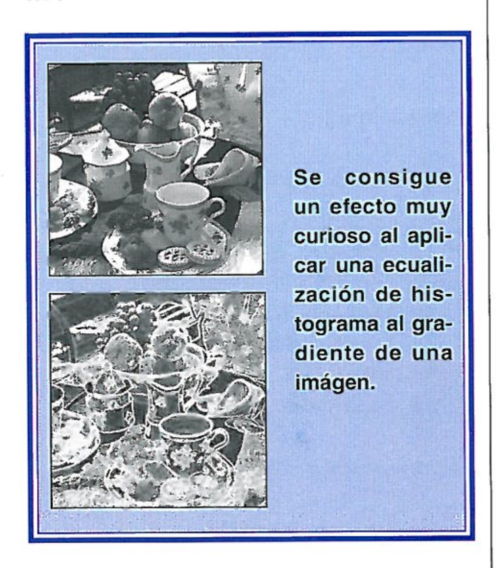
donde :

x, y es el punto en el que queremos calcular el valor de la gaussiana.

x0, y0 es el punto en el que está centrada la gaussiana

k es una constante de normalización.

kx, ky nos controlan cómo disminuye el valor de la función al alejarnos de su centro, y lo normal es que valgan lo mismo.



Se consigue un efecto muy curioso al aplicar una ecualización de histograma al gradiente de una imágen.

En el caso discreto, que es el que interesa, no es posible calcular k a priori. Lo que haremos entonces es rellenar la máscara sin tener en cuenta k, sumar después el valor de todos sus elementos e igualar k a la inversa de dicha suma.

MÉTODOS DE ELIMINACIÓN DEL RUIDO

Eliminar el ruido o, al menos, reducir su influencia es algo bastante deseable en la mayoría de las ocasiones. El ruido puede provenir de fuentes muy diversas : ruido eléctrico en sensores, granularidad de las películas fotográficas, errores o interferencias en la transmisión de la imagen, etc.

Aunque existen muchos tipos de ruido, hay dos que suelen aparecer con mucha frecuencia : el ruido impulsivo, que consiste en puntos aleatoriamente distribuidos por la imagen con niveles de gris muy diferentes de los de sus vecinos, y el ruido uniforme o gaussiano, que tiene media nula, y está distribuido de forma homogénea por toda la imagen.

Una forma muy rudimentaria de eliminar ruido es utilizar un filtro de suavizado. Como su efecto es, básicamente, sustituir el valor de un punto por la media de los valores de los pixels cercanos, los puntos con valores extraños verán reducida su influencia. El problema está en que, además de suavizar el ruido emborronamos la imagen.

El método de la media restringida se basa en la idea antes expuesta, pero sólo suaviza un punto si éste se diferencia poco de sus vecinos. Así se consigue que los contornos de las figuras que aparecen en la imagen no se emborronen, mientras que las zonas granulosas, al suavizarse, se vuelven uniformes. Dependiendo del nivel de ruido que tenga la imagen habrá que especificar un umbral a partir del cual se considerará que la diferencia del nivel de gris respecto a la media local no es consecuencia del ruido y no se suavizará. La determinación de un umbral adecuado, por tanto, hará que esta técnica sea más o menos eficiente.

La media restringida elimina bastante bien el ruido gaussiano, pero deja sin cambios al impulsivo. Para eliminar este último el método más adecuado es el filtrado de mediana, que consiste en sustituir el valor de un punto por la mediana de los valores de los pixels de una ventana centrada en él. La mediana es el valor que está en medio de la lista ordenada de los valores de los puntos de la ventana. Por ejemplo, si en la ventana están los niveles de gris siguientes : {1, 7, 15, 24, 32, 2, 5, 5, 12} la mediana será el valor central de {1, 2, 5, 5, 7, 12, 15, 24, 32}, es decir, el 7.

En general los resultados del filtrado de mediana son muy superiores a los del filtrado de media restringida.

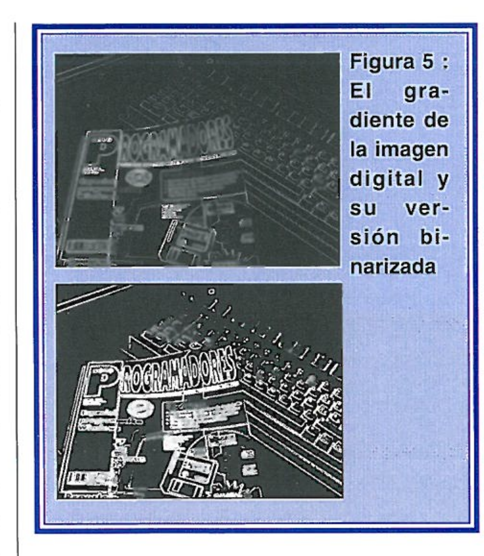


Figura 5 : El gradiente de la imagen digital y su versión binarizada

MÉTODOS DE EXTRACCIÓN DE BORDES

Un borde es una línea que separa dos zonas de la imagen que tienen niveles de gris suficientemente distintos, esto es, se diferencian en más de un cierto valor umbral.

El gradiente de una función es básicamente su derivada, y nos mide cuanto ha crecido o decrecido el valor de dicha función en un punto determinado. Podemos usar la magnitud del gradiente para medir la variación del nivel de gris y, por tanto, para detectar la presencia de bordes en la imagen.

Un método bastante burdo para calcular el gradiente consiste en ir restando los valores de cada par de puntos adyacentes. A mayor diferencia, mayor será el gradiente y más destacado será el borde.

Sin embargo no suele usarse directamente la diferencia como método para la extracción de gradientes porque es demasiado sensible a las variaciones de luminancia de la imagen, y no discrimina muy bien los contornos. Lo más corriente es usar una máscara bidimensional que, además de obtener la diferencia en la dirección del gradiente, promedie en la dirección normal al mismo. Una de las más utilizadas es la Sobel :

Para el gradiente según el eje X:

$$Gx = 1/4 * \begin{vmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{vmatrix}$$

Para el gradiente según el eje Y

$$Gy = 1/4 * \begin{vmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{vmatrix}$$

Después, para obtener la imagen de bordes, se calcula la norma del gradiente :

G = sqrt (Gx^2 + Gy^2)

Que puede aproximarse, a efectos de economía de tiempo de proceso, por la suma de las magnitudes de los gradientes según los ejes :

G = abs (Gx) + abs (Gy)

Una vez obtenido el gradiente, se suele umbralizar o binarizar la imagen, para escoger únicamente los bordes que estén mejor diferenciados. El proceso de binarización consiste en poner al máximo valor posible los pixels que tengan un nivel de gris mayor que un cierto umbral, y dejar el resto a cero.

MEJORA DE CONTRASTE

Muchas veces una iluminación defectuosa, o un mal procedimiento de digitalización, pueden ser responsables de que las imágenes no tengan suficiente contraste. El que una imagen tenga bajo contraste quiere decir que los niveles de gris que la componen no se pueden distinguir fácilmente.

También puede ser que haya zonas de la imagen que contengan información que nos interesa y que no seamos capaces de ver porque son, en conjunto, demasiado claras u oscuras. Para aumentar la discernibilidad de dichas imágenes o zonas de imágenes, lo que tenemos que hacer es separar los niveles de gris que las componen. A veces podemos conseguir esto con una sencilla normalización, pero no suele ser el caso. Algo que nos será muy útil para comprender mejor todo esto es el concepto de histograma.

El histograma de una imagen es una función que indica cuantos puntos tiene aquella en cada nivel de gris. En una imagen oscura, por ejemplo, los niveles de gris más bajos (cercanos al negro) serán los más ocupados, y el histograma presentará una cresta en su zona inferior. Por el contrario, el histograma de una imagen clara tendrá una cresta en sus niveles altos.

Observando el histograma de una imagen es fácil saber que técnica debemos emplear para mejorarla, pero, básicamente, todas consisten en redistribuir los puntos de los niveles más llenos sobre las zonas de los niveles menos ocupados, para conseguir que los primeros estén más separados y sean más fácilmente distinguibles.

La ecualización es una técnica que intenta ajustar el histograma de una imagen a una función de igual valor para todos sus niveles de gris. Esta técnica es bastante útil para mejorar imágenes de muy mala calidad, pero no suele ser adecuada en la mayoría de las ocasiones, ya que no es ni mucho menos normal que el histograma de una imagen sea una función constante. En general la técnica de mejora de contraste debe ser aplicada de forma particular para cada imagen.

Un método mucho más flexible para el ajuste de histogramas es el de la rampa, que consiste en saturar un cierto número de niveles de gris hacia el blanco o el negro, "estirando" el resto para ocupar todo el rango disponible. El problema de este método es, que no sabemos, para una imagen arbitraria, cual es el número óptimo de niveles de gris que debemos saturar hacia el negro o blanco para obtener un buen resultado. Una técnica para solucionar esto, consiste en saturar hacia el negro y hacia el blanco, un pequeño porcentaje del número total de puntos de la imagen. Así se garantiza que la saturación no se llevará a cabo, si los niveles cercanos al límite están bastante llenos, y que se encontrará un nivel adecuado, a partir del cual saturar si hay muchos niveles poco llenos.

Ajustando los porcentajes de saturación de forma adecuada es fácil mejorar cualquier imagen utilizando el método de la rampa. Sin embargo siempre pueden construirse funciones a medida : puede aplicarse un logaritmo a los niveles de gris, para aumentar el contraste de las zonas oscuras, una parábola o exponencial para las claras, etc.

CÓDIGO DEL DISCO

Para usar las rutinas que acompañan a este artículo pueden hacerse dos cosas. Una, crear una librería con ellas y, la otra, añadirle al listado de las rutinas una función main y compilar el programa normalmente. Como ejemplo de este último método puede añadirse el siguiente trozo de código al fichero de rutinas que se encuentra impreso en las páginas siguientes:

```
/* Objetivo : crear un programa que lea el fichero "imgin.raw" del disco, le haga un suavizado gaussiano, extraiga su gradiente y luego guarde el resultado en el fichero "imgout.raw". */

main ()
{
imagen img;
fimagen fimg, fimgout;
int rad = 3;

/* reservamos memoria para las imágenes necesarias */

crearimagen (&img, defimgsizey, defimgsizex);
crearfimagen (&fimg, defimgsizey, defimgsizex);
crearfimagen (&fimgout, defimgsizey, defimgsizex);

/* leemos el fichero del disco */

cargar imagen ("imgin.raw", img);

/* realizamos operaciones */

suavizado_gauss (img, rad);
imagen2fimagen (img, fimg);
gradientef (fimg, fimgout);
normalizarf (fimgout, 0.0, (float) (defnumgrises - 1));
fimagen2imagen (fimgout, img);

/* guardamos resultado */

salvarimagen ("imgout.raw", img);

/* liberamos memoria */

destruirfimagen (&fimgout);
destruirfimagen (&fimg);
destruirimagen (&img);
}
```



RUTINAS.C

```
/*******************
* CONSTANTES *
*******************/

#define defimgsizey 512
#define defimgsizex 512
#define defnumgrises 256

#ifndef MAXFLOAT
#   define MAXFLOAT 3.402823466e+38f
#endif

/*******
* TIPOS *
*******/

typedef struct {
  int sizey, sizex;
  unsigned char **dat;
} imagen;

typedef struct {
  int sizey, sizex;
  float **dat;
} fimagen;

/*******************
* FUNCIONES BASICAS *
*******************/

/************************************************************************

  crearimagen :

   RESERVA MEMORIA PARA ALMACENAR UNA IMAGEN DE TIPO BYTE DE
   TAMAÑO
        imgsizey x imgsizex.

************************************************************************/

void crearimagen (imagen *img, int imgsizey, int imgsizex)
{
  int k;

  /* COMPROBAMOS QUE LAS DIMENSIONES SEAN VALIDAS */

  if ((imgsizex <= 0) || (imgsizey <= 0)) {
    fprintf (stderr, "\nCREARIMAGEN : parámetros no validos.\n");
    exit (1);
  }

  img->sizey = imgsizey;
  img->sizex = imgsizex;

  /* RESERVAMOS MEMORIA */

  img->dat = (unsigned char **) malloc (sizeof (unsigned char *) * img->sizey);
  if (img->dat == NULL) {
    fprintf (stderr, "\nCREARIMAGEN : error al reservar memoria.\n");
    exit (1);
  }

  img->dat [0] = (unsigned char *) malloc (sizeof (unsigned char) * img->sizex * img-
>sizey);
  if (img->dat [0] == NULL) {
    fprintf (stderr, "\nCREARIMAGEN : error al reservar memoria.\n");
    exit (1);
  }

  for (k = 1; k < img->sizey; k ++)
    img->dat [k] = img->dat [k - 1] + img->sizex;
}

/************************************************************************

  crearfimagen :

   RESERVA MEMORIA PARA ALMACENAR UNA IMAGEN DE TIPO FLOAT DE
TAMAÑO
        imgsizey x imgsizex.

************************************************************************/

void crearfimagen (fimagen *img, int imgsizey, int imgsizex)
{
  int k;

  /* COMPROBAMOS QUE LAS DIMENSIONES SEAN VALIDAS */

  if ((imgsizex <= 0) || (imgsizey <= 0)) {
    fprintf (stderr, "\nCREARFIMAGEN : parámetros no validos.\n");
    exit (1);
  }

  img->sizey = imgsizey;
  img->sizex = imgsizex;

  /* RESERVAMOS MEMORIA */

  img->dat = (float **) malloc (sizeof (float *) * img->sizey);
  if (img->dat == NULL) {
    fprintf (stderr, "\nCREARFIMAGEN : error al reservar memoria.\n");
    exit (1);
  }

  img->dat [0] = (float *) malloc (sizeof (float) * img->sizex * img->sizey);
  if (img->dat [0] == NULL) {
    fprintf (stderr, "\nCREARFIMAGEN : error al reservar memoria.\n");
    exit (1);
  }

  for (k = 1; k < img->sizey; k ++)
    img->dat [k] = img->dat [k - 1] + img->sizex;
}

/************************************************************************

  destruirimagen :

   LIBERA LA MEMORIA RESERVADA PARA UNA IMAGEN.

************************************************************************/

void destruirimagen (imagen *img)
{
  free (img->dat [0]);
  free (img->dat);

  img->dat = NULL;
  img->sizey = 0;
  img->sizex = 0;
}

/************************************************************************

  destruirfimagen :

   LIBERA LA MEMORIA RESERVADA PARA UNA FIMAGEN.

************************************************************************/

void destruirfimagen (fimagen *img)
{
  free (img->dat [0]);
  free (img->dat);

  img->dat = NULL;
  img->sizey = 0;
  img->sizex = 0;
}

/************************************************************************

  imagen2fimagen :

   CONVIERTE UNA IMAGEN DE TIPO BYTE A IMAGEN DE REALES. LAS DOS
   DEBEN TENER EL MISMO TAMAÑO.

************************************************************************/

void imagen2fimagen (imagen img, fimagen fimg)
{
  int y, x;

  for (y = 0; y < img.sizey; y ++)
    for (x = 0; x < img.sizex; x ++)
      fimg.dat [y][x] = (float) img.dat [y][x];
}

/************************************************************************

  fimagen2imagen :

   CONVIERTE UNA IMAGEN DE REALES A IMAGEN DE TIPO BYTE. CONVIENE
   NORMALIZAR AL RANGO (0.. NUMGRISES - 1) ANTES DE HACER LA CONVERSION
   PARA QUE LA PERDIDA DE INFORMACION SEA LA MINIMA POSIBLE. LAS DOS
   IMAGENES DEBEN TENER EL MISMO TAMAÑO.

************************************************************************/

void fimagen2imagen (fimagen fimg, imagen img)
{
  int y, x;

  for (y = 0; y < img.sizey; y ++)
    for (x = 0; x < img.sizex; x ++)
      img.dat [y][x] = (unsigned char) fimg.dat [y][x];
}

/************************************************************************

  cargarimagen :

   LEE UN FICHERO RAW DE TIPO BYTE. LA VARIABLE DE TIPO IMAGEN DONDE
   SE CARGA EL FICHERO DEBE HABER SIDO INICIALIZADA CON CREARIMAGEN
   PREVIAMENTE.

************************************************************************/

void cargarimagen (nombre, img)
     char *nombre;
     imagen img;
{
  FILE *stream;

  stream = fopen (nombre, "rb");
  if (stream == NULL) {
    printf ("\nCARGARIMAGEN : No puedo leer el fichero '%s'.\n", nombre);
    exit (1);
  }
  else
    fread ((unsigned char *) img.dat [0], sizeof (unsigned char), img.sizex * img.sizey, stre-
am);

  fclose (stream);
}

/************************************************************************

  salvarimagen :

   SALVA UNA IMAGEN AL DISCO EN FORMATO RAW CON DATOS BYTE.

************************************************************************/

void salvarimagen (nombre, img)
     char *nombre;
     imagen img;
{
  FILE *stream;

  stream = fopen (nombre, "wb");
  if (stream == NULL) {
    printf ("\nSALVARIMAGEN : No puedo escribir el fichero '%s'.\n", nombre);
    exit (1);
  }
  else
    fwrite ((unsigned char *) img.dat [0], sizeof (unsigned char), img.sizex * img.sizey, stre-
am);

  fclose (stream);
}

/************************************************************************

  rellenarf :

   RELLENA UNA FIMAGEN CON UN VALOR CONSTANTE CUALQUIERA.

************************************************************************/

void rellenarf (fimagen fimg, float val)
{
```

```c
  int y, x;

  for (y = 0; y < fimg.sizey; y ++)
    for (x = 0; x < fimg.sizex; x ++)
      fimg.dat [y][x] = val;
}

/****************************************************************************

  copiar :

    COPIA EL CONTENIDO DE UNA IMAGEN EN OTRA. AMBAS DEBEN SER DEL
MISMO
    TAMAÑO.

****************************************************************************/

void copiar (imagen img, imagen img2)
{
  memcpy ((unsigned char *) img2.dat [0], (unsigned char *) img.dat [0],
          img.sizey * img.sizex);
}

/****************************************************************************

  binarizar :

    BINARIZA UNA IMAGEN EN FUNCION DEL NIVEL DE GRIS ESPECIFICADO POR
    UMBRAL.

****************************************************************************/

void binarizar (imagen img, unsigned char umbral)
{
  int y, x;

  for (y = 0; y < img.sizey; y ++)
    for (x = 0; x < img.sizex; x ++)
      if (img.dat [y][x] > umbral)
              img.dat [y][x] = defnumgrises - 1;
      else
              img.dat [y][x] = 0;
}

/****************************************************************************

  truncarf :

    TRUNCA LOS VALORES DE UNA FIMAGEN PARA QUE NO SALGAN DEL RANGO
    ESPECIFICADO POR LOS LIMITES V0 Y VF.

****************************************************************************/

void truncarf (fimagen img, float v0, float vf)
{
  int y, x;

  for (y = 0; y < img.sizey; y ++)
    for (x = 0; x < img.sizex; x ++)
      if (img.dat [y][x] < v0)
              img.dat [y][x] = v0;
      else
              if (img.dat [y][x] > vf)
                img.dat [y][x] = vf;
}

/****************************************************************************

  normalizarf :

    NORMALIZA EL RANGO DE VALORES DE UNA FIMAGEN A LOS VALORES LIMITE
    QUE SE ESPECIFIQUEN EN FIXEDMIN Y FIXEDMAX.

****************************************************************************/

void normalizarf (fimagen fimg, float fixedmin, float fixedmax)
{
  int y, x;
  float max, min, coef;

  /* BUSQUEDA DE LOS VALORES MAXIMO Y MINIMO DE LA FIMAGEN */

  max = - MAXFLOAT;
  min = MAXFLOAT;
  for (y = 0; y < fimg.sizey; y ++)
    for (x = 0; x < fimg.sizex; x ++) {
      if (min > fimg.dat [y][x])
              min = fimg.dat [y][x];
      if (max < fimg.dat [y][x])
              max = fimg.dat [y][x];
    }

  /* CALCULO DEL COEFICIENTE DE NORMALIZACION */

  if (max != min)
    coef = (fixedmax - fixedmin) / (max - min);
  else
    coef = 1.0;

  /* APLICACION DE LA TRANSFORMACION LINEAL DE NORMALIZACION */

  for (y = 0; y < defimgsizey; y ++)
    for (x = 0; x < defimgsizex; x ++)
      fimg.dat [y][x] = coef * (fimg.dat [y][x] - min) + fixedmin;
}

/****************************************************************************

  prod_convolucionf :

    CALCULA EL PRODUCTO DE CONVOLUCION DE UNA MASCARA CON UNA
    FIMAGEN. LA MASCARA DEBE SER MAS PEQUEÑA QUE LA FIMAGEN Y TENER
    DIMENSIONES IMPARES. LOS BORDES DE FIMGOUT DESDE 0 A MASKSIZEYX / 2
    SE RELLENAN CON CEROS. FIMG Y FIMGOUT DEBEN SER DE IGUAL TAMAÑO.

****************************************************************************/

void prod_convolucionf (fimagen fimg, fimagen mask, fimagen fimgout)
{
  int y, x, dy, dx;
  int masksizey2, masksizex2;

  masksizey2 = mask.sizey / 2;
  masksizex2 = mask.sizex / 2;

  /* PUESTA A CERO DE LA IMAGEN DE SALIDA */

  rellenarf (fimgout, 0.0);

  /* CALCULO DEL PRODUCTO DE CONVOLUCION */

  for (y = masksizey2; y < fimg.sizey - masksizey2; y ++)
    for (x = masksizex2; x < fimg.sizex - masksizex2; x ++)
      for (dy = - masksizey2; dy <= masksizey2; dy ++)
        for (dx = - masksizex2; dx <= masksizey2; dx ++)
          fimgout.dat [y][x] += fimg.dat [y + dy][x + dx] * mask.dat [masksizey2 +
                           dy][masksizex2 + dx];
}

/************************
* FUNCIONES DE SUAVIZADO *
************************/

/****************************************************************************

  suavizado_media :

    SUAVIZA UNA IMAGEN POR EL METODO DE LA MEDIA 3X3. NUMPASADAS
    ESPECIFICA EL NUMERO DE VECES QUE DEBE APLICARSE EL FILTRO A LA
    IMAGEN. TAMBIEN PUEDE CONSEGUIRSE EL MISMO EFECTO CONVOLUCIONAN-
DO
    LA IMAGEN DE ENTRADA CON UNA MASCARA 3X3 DE VALOR CONSTANTE 1/9.

****************************************************************************/

void suavizado_media (imagen img, int numpasadas)
{
  imagen imgaux;
  int x, y, dx, dy, p;
  float media, k;

  k = 1.0 / 9.0;

  crearimagen (&imgaux, img.sizey, img.sizex);

  for (p = 0; p < numpasadas; p ++) {

    copiar (img, imgaux);

    /* SUSTITUCION DEL VALOR DE CADA PUNTO POR LA MEDIA LOCAL 3X3 */

    for (y = 1; y < img.sizey - 1; y ++)
      for (x = 1; x < img.sizex - 1; x ++) {
              media = 0.0;
              for (dy = - 1; dy < 2; dy ++)
                for (dx = - 1; dx < 2; dx ++)
                  media += (float) imgaux.dat [y + dy][x + dx];
              img.dat [y][x] = (unsigned char) (media * k);
      }
  }

  destruirimagen (&imgaux);
}

/****************************************************************************

  suavizado_gauss :

    SUAVIZA UNA IMAGEN CONVOLUCIONANDOLA CON UNA GAUSSIANA DE UN
    RADIO DADO.LA GAUSSIANA SE NORMALIZA A 1 PARA QUE EL VALOR MEDIO DE
    LA IMAGEN CON LA QUE SE CONVOLUCIONA NO VARIE.

****************************************************************************/

void suavizado_gauss (imagen img, int radio)
{
  fimagen fimg, mask, fimgout;
  int masksize;
  double kalpha;
  float normcoef, sum;
  int y, x;

  masksize = 2 * radio + 1;
  kalpha = log (0.1) / ((double) radio * (double) radio);

  crearfimagen (&mask, masksize, masksize);

  /* RELLENAMOS MASCARA CON GAUSSIANA Y HALLAMOS EL COEFICIENTE DE
     NORMALIZACION */

  sum = 0.0;
  for (y = - radio; y <= radio; y ++)
    for (x = - radio; x <= radio; x ++) {
      mask.dat [y + radio][x + radio] = (float) exp (kalpha * (double) (x * x + y * y));
      sum += mask.dat [y + radio][x + radio];
    }

  if (sum != 0.0)
    normcoef = 1.0 / sum;
  else
    normcoef = 1.0;

  for (y = 0; y < masksize; y ++)
    for (x = 0; x < masksize; x ++)
      mask.dat [y][x] *= normcoef;

  /* CONVOLUCIONAMOS */

  crearfimagen (&fimg, img.sizey, img.sizex);
  crearfimagen (&fimgout, img.sizey, img.sizex);

  imagen2fimagen (img, fimg);
  prod_convolucionf (fimg, mask, fimgout);
  fimagen2imagen (fimgout, img);

  destruirfimagen (&fimgout);
  destruirfimagen (&fimg);
  destruirfimagen (&mask);
}

/**********************************
* FUNCIONES DE ELIMINACION DE RUIDO *
**********************************/

/****************************************************************************

  media_rest :

    FILTRA UNA IMAGEN UTILIZANDO EL METODO DE LA MEDIA RESTRINGIDA PARA
    LA ELIMINACION DE RUIDO. NUMSUAV INDICA EL NUMERO DE PASADAS PARA
EL
    DIFERENCIA, EN NUMERO DE NIVELES DE GRIS, QUE SE PUEDE
CONSIDERAR
```

```
  COMO RUIDO.
*************************************************************************/

void media_rest (imagen img, int numsuav, int umbral)
{
  imagen softimg;
  int y, x;

  crearimagen (&softimg, img.sizey, img.sizex);

  copiar (img, softimg);
  suavizado_media (softimg, numsuav);

  for (y = 0; y < img.sizey; y ++)
   for (x = 0; x < img.sizex; x ++)
    if (abs ((double) img.dat [y][x] - (double) softimg.dat [y][x]) < (double) umbral)
      img.dat [y][x] = softimg.dat [y][x];

  destruirimagen (&softimg);
}

/*************************************************************************

  filtrado_mediana :

   PASA UN FILTRADO DE MEDIANA A UNA IMAGEN PARA ELIMINACION DE RUIDO.
   WSIZE INDICA EL TAMANIO DE LA VENTANA EN LA QUE SE CALCULA EL VALOR
   DE LA MEDIANA (NORMALMENTE 3 ES SUFICIENTE). LA PARTE DE ORDENA-
CION
   Y BUSQUEDA DE LA MEDIANA PUEDE OPTIMIZARSE PARA ACELERAR LA RUTI-
NA.

*************************************************************************/

void filtrado_mediana (imagen img, int wsize)
{
  imagen img2, vect;
  int vsize2, vsize, dvsize, medianindex;
  int y, x, dy, dx;
  int k, l;
  unsigned char vtemp;

  dvsize = (wsize - 1) / 2;
  vsize = dvsize * 2 + 1;
  vsize2 = vsize * vsize;
  medianindex = vsize2 / 2 + 1;

  crearimagen (&img2, img.sizey, img.sizex);
  crearimagen (&vect, vsize2, 1);

  copiar (img, img2);

  for (y = dvsize + 1; y < img.sizey - dvsize - 1; y ++)
   for (x = dvsize + 1; x < img.sizex - dvsize - 1; x ++) {

    /* RELLENADO DEL VECTOR CON LOS VALORES DE LA VENTANA */

    for (dy = -dvsize; dy <= dvsize; dy ++)
           for (dx = -dvsize; dx <= dvsize; dx ++)
            vect.dat [(dvsize + dy) * vsize + dx + dvsize][0] = img2.dat [y + dy][x + dx];

    /* ORDENACION DEL VECTOR */

    for (k = 0; k < vsize2 - 1; k ++)
           for (l = 1; l < vsize2; l ++)
            if (vect.dat [l][0] < vect.dat [k][0]) {
             vtemp = vect.dat [k][0];
             vect.dat [k][0] = vect.dat [l][0];
             vect.dat [l][0] = vtemp;
            }

    /* OBTENCION DE LA MEDIANA */

    img.dat [y][x] = vect.dat [medianindex][0];
   }

  destruirimagen (&vect);
  destruirimagen (&img2);
}

/***********************************
* FUNCIONES DE EXTRACCION DE BORDES *
***********************************/

/*************************************************************************

  gradientef :

   EXTRAE EL GRADIENTE DE UNA FIMAGEN MEDIANTE EL METODO DE SOBEL.

*************************************************************************/

void gradientef (fimagen fimg, fimagen fimgout)
{
  int masksize = 3;
  int sobelxmask [3][3] = {1, 0, -1, 2, 0, -2, 1, 0, -1};
  int sobelymask [3][3] = {-1, -2, -1, 0, 0, 0, 1, 2, 1};

  fimagen mask, sobximg, sobyimg;
  int y, x;

  crearfimagen (&mask, masksize, masksize);
  crearfimagen (&sobximg, fimg.sizey, fimg.sizex);
  crearfimagen (&sobyimg, fimg.sizey, fimg.sizex);

  /* CALCULO DEL GRADIENTE EN X */

  for (y = 0; y < masksize; y ++)
   for (x = 0; x < masksize; x ++)
    mask.dat [y][x] = 0.25 * (float) sobelxmask [y][x];

  prod_convolucionf (fimg, mask, sobximg);

  /* CALCULO DEL GRADIENTE EN Y */

  for (y = 0; y < masksize; y ++)
   for (x = 0; x < masksize; x ++)
    mask.dat [y][x] = 0.25 * (float) sobelymask [y][x];

  prod_convolucionf (fimg, mask, sobyimg);

  /* CALCULO DEL MODULO DEL GRADIENTE */

  for (y = 0; y < fimg.sizey; y ++)
   for (x = 0; x < fimg.sizex; x ++)
    fimgout.dat [y][x] = (float) sqrt ((double) (sobximg.dat [y][x] * sobximg.dat
[y][x]
                       + sobyimg.dat [y][x] * sobyimg.dat [y][x]));

  destruirfimagen (&sobyimg);
  destruirfimagen (&sobximg);
  destruirfimagen (&mask);
}

/***********************************
* FUNCIONES DE MEJORA DE CONTRASTE *
***********************************/

/*************************************************************************

  ecualizar_hist :

   MEJORA EL CONTRASTE DE UNA IMAGEN POR EL METODO DE ECUALIZACION
DE
   HISTOGRAMA.

*************************************************************************/

void ecualizar_hist (imagen img)
{
  fimagen hist, nuevohist;
  float numimgpoints;
  int y, x, k;

  numimgpoints = (float) img.sizey * (float) img.sizex;

  crearfimagen (&hist, defnumgrises, 1);
  crearfimagen (&nuevohist, defnumgrises, 1);

  rellenarf (hist, 0.0);
  rellenarf (nuevohist, 0.0);

  /* CALCULAMOS HISTOGRAMA */

  for (y = 0; y < img.sizey; y ++)
   for (x = 0; x < img.sizex; x ++)
    hist.dat [(int) img.dat [y][x]][0] += 1.0;

  /* ECUALIZAMOS */

  nuevohist.dat [0][0] = hist.dat [0][0];
  for (k = 1; k < defnumgrises; k ++)
   nuevohist.dat [k][0] = nuevohist.dat [k - 1][0] + hist.dat [k][0];

  for (k = 0; k < defnumgrises; k ++)
   nuevohist.dat [k][0] = nuevohist.dat [k][0] * (float) (defnumgrises - 1) / numimgpoints;

  /* APLICAMOS NUEVO HISTOGRAMA */

  for (y = 0; y < img.sizey; y ++)
   for (x = 0; x < img.sizex; x ++)
    img.dat [y][x] = (unsigned char) nuevohist.dat [(int) img.dat [y][x]][0];
}

/*************************************************************************

  rampa_hist :

   REALIZA UNA MEJORA DE CONTRASTE APLICANDO UNA RAMPA AL HISTOGRA-
MA
   DE LA IMAGEN DE ENTRADA. PCNEGRO Y PCBLANCO INDICAN EL PORCENTAJE
   DE PUNTOS DE LA IMAGEN QUE SE SATURARAN HACIA EL NEGRO Y EL BLAN-
CO,
   RESPECTIVAMENTE (UN 1% PARA AMBOS SUELE SER SUFICIENTE).

*************************************************************************/

void rampa_hist (imagen img, float pcnegro, float pcblanco)
{
  fimagen fimg, hist;
  float numpuntosimg, puntosanegro, puntosablanco, numpuntos;
  int nuevonegro, nuevoblanco;
  int y, x, k;

  crearfimagen (&hist, defnumgrises, 1);
  crearfimagen (&fimg, img.sizey, img.sizex);
  imagen2fimagen (img, fimg);

  numpuntosimg = (float) img.sizey * (float) img.sizex;
  puntosanegro = numpuntosimg * 0.01 * pcnegro;
  puntosablanco = numpuntosimg * 0.01 * pcblanco;

  /* CALCULAMOS HISTOGRAMA */

  rellenarf (hist, 0.0);

  for (y = 0; y < img.sizey; y ++)
   for (x = 0; x < img.sizex; x ++)
    hist.dat [(int) img.dat [y][x]][0] += 1.0;

  /* CALCULAMOS NUEVOS LIMITES PARA EL HISTOGRAMA */

  numpuntos = 0.0;
  k = 0;
  while ((k < defnumgrises - 1) && (numpuntos < puntosanegro)) {
   numpuntos += hist.dat [k][0];
   k ++;
  }
  nuevonegro = k;

  numpuntos = 0.0;
  k = defnumgrises - 1;
  while ((k > 0) && (numpuntos < puntosablanco)) {
   numpuntos += hist.dat [k][0];
   k --;
  }
  nuevoblanco = k;

  /* TRUNCAMOS GRISES QUE CAIGAN FUERA DE LOS NUEVOS LIMITES */

  truncarf (fimg, (float) nuevonegro, (float) nuevoblanco);

  /* USAMOS UNA NORMALIZACION PARA EL ESTIRADO DE LOS NIVELES DE GRIS
*/

  normalizarf (fimg, 0.0, (float) (defnumgrises - 1));

  fimagen2imagen (fimg, img);

  destruirfimagen (&fimg);
  destruirfimagen (&hist);
}
```

MODO ALFANUMÉRICO

Sergio Ríos

En esta entrega del curso de programación de la VGA abordaremos los aspectos más importantes de los modos de texto, así como las bases fundamentales de la generación de caracteres, que, por extensión, son también aplicables a los modos gráficos.

En cierta medida, el estudio de los modos de texto está considerado como uno de los temas menos atractivos de la programación de la VGA.

En realidad, lo más interesante de los modos de texto no es quizás la presentación de texto por pantalla propiamente dicha, sino el proceso de generación de caracteres, la utilización de distintos tipos de fuentes de texto, etc.

MODOS ALFANUMÉRICOS Y MODOS GRÁFICOS

En los modos gráficos, dicha presentación se hace punto a punto, a partir de la matriz de puntos que representa gráficamente cada uno de los caracteres. En cambio, en los modos alfanuméricos, dicha presentación se hace carácter a carácter: simplemente se indica el código del carácter que se desea mostrar, así como su atributo, y es el propio hardware de la tarjeta gráfica, el que se encarga de mostrar dicho carácter de acuerdo con la información almacenada en la matriz de puntos que conforman el carácter.

Los modos alfanuméricos son bastante más rápidos que los modos gráficos a la hora de representar texto, porque en los primeros ésta operación se realiza enteramente por hardware. Sin embargo, para que esto sea posible, existen ciertas limitaciones, siendo una de las más importantes el hecho de que en los modos alfanuméricos, sólo resulta posible mostrar caracteres en unas determinadas posiciones prefijadas. En estos modos, la pantalla queda virtualmente dividida en filas y columnas de celdillas en las que cabe un carácter de la fuente que se esté usando. En cambio, en los modos gráficos, es perfectamente factible, situar los caracteres en cualquier punto de la pantalla (aunque para ello no podamos, en general, acudir a las rutinas de la ROM BIOS).

En los modos alfanuméricos, se accede a la memoria de vídeo de la tarjeta VGA en las direcciones B8000h-BFFFFh del espacio de direcciones de la CPU. En esta zona de memoria aparecerán alternados los códigos ASCII de los caracteres con sus correspondientes atributos. En realidad, lo que sucede es bastante más complejo: en la propia VGA, la memoria se organiza en cuatro "planos" (siguiendo la terminología original de IBM "bit-planes") de 64KB. En el plano 1 se almacenan los códigos ASCII de los caracteres que aparecen en pantalla; En el plano 2 se guardan sus atributos, y el plano 3 se utiliza para almacenar las matrices de puntos, que conforman los caracteres. Como ya hemos comentado, en los modos alfanuméricos, el hardware controla la generación de caracteres; pues bien, esto lo hace accediendo indirectamente al plano 3. Todo este proceso se realiza de forma transparente para el programador.

FUENTES DE LA VGA

En la ROM BIOS de la tarjeta VGA, están almacenados las fuentes o conjuntos de caracteres estándar de los modos alfanuméricos, con las matrices de puntos correspondientes. Cada uno de los conjuntos se compone de 256 caracteres. En la VGA puede haber dos de estos conjuntos activos al mismo tiempo, lo que posibilita trabajar con conjunto de caracteres virtual de hasta 512 caracteres.

La ROM BIOS de la VGA estándar dispone de 5 juegos de caracteres, cada uno con un determinado tamaño de carácter: 8x8 puntos, 8x14 puntos,

TABLA 1	
Ejemplos de atributos en modo monocromo	
Código	Atributo
07h	Normal
0Fh	Intenso
01h	Subrayado
09h	Subrayado intenso
70h	Vídeo inverso
F0h	Vídeo inverso y parpadeo

8x16 puntos, 9x14 puntos y 9x16 puntos. Estas dos últimas fuentes son suplementarias. Por supuesto, es posible utilizar fuentes definidas por el usuario en lugar de las fuentes estándar.

En la VGA, cada plano de bits consta de 8 bloques de 8KB. En cada uno de ellos caben los patrones (matrices de puntos) de un conjunto de 256 caracteres. Por ello, es posible cargar hasta un máximo de 8 fuentes en la memoria de la VGA (plano 3).

Veamos ahora cómo son las matrices de puntos de los caracteres. En principio, cada carácter se compone de tantas celdillas como indica la resolución de su fuente. Por ejemplo, en una fuente de 8x14, un carácter consta de 8x14 (112) puntos. Cada punto se representa mediante un pixel. En memoria, la matriz de puntos se almacena en bytes consecutivos, y cada byte se corresponde con una fila, de la matriz de puntos del carácter en cuestión. En cualquier caso, el bit más significativo hace referencia al punto situado más a la izquierda en cada fila de la matriz.

Siguiendo este esquema, la pantalla queda organizada como un mosaico de celdas, cada una del tamaño de un carácter para la fuente usada, y sin separación entre ellas. Esta ausencia de separación obliga, a que el tamaño efectivo de los caracteres sea algo menor al que viene dado por su fuente, a fin de lograr una cierta separación entre los caracteres mostrados en pantalla, tanto vertical como horizontalmente.

La fuente de 8x8 puntos es la más pequeña de todas. Los caracteres tienen un tamaño efectivo de 7x7 puntos. Según el número de líneas de barrido generadas por la tarjeta, se pueden conseguir 25, 43, 50 o 60 líneas de texto de 80 columnas. Los caracteres que usan la fuente de 8x14 puntos tienen un tamaño efectivo de 7x9 puntos. Las dos filas superiores y las tres inferiores están en blanco, salvo en el caso de los caracteres gráficos, para evitar discontinuidades entre caracteres adyacentes.

LAS FUENTES SUPLEMENTARIAS, O COMO SEPARAR CARACTERES

Las fuentes de 9x14 puntos y 9x16 puntos se denominan fuentes suplementarias, puesto que toman como base las matrices de puntos de la fuente de 8x14 puntos. A cada fila de la fuente 8x14 se añade un noveno bit, que actúa como un simple separador de caracteres (está siempre en blanco). Sólo los caracteres que se beneficiarían al pasar a 9x14 puntos se redefinen (se una una nueva matriz de puntos). Cada una de las matrices de puntos redefinidas de los caracteres de la fuente suplementaria va precedida del código ASCII del carácter en cuestión. El código ASCII 00h indica el final del conjunto de caracteres suplementarios.

Los modos alfanuméricos son cuatro: 0, 1, 2, 3 y 7. Los modos 0, 1, 2 y 3 son modos de 16 colores, y el modo 7 es monocromo.

TABLA 2	
Colores por defecto de la paleta de 16 colores	
Registro de paleta	Color
0	Negro
1	Azul
2	Verde
3	Cían
4	Rojo
5	Rosa
6	Marrón
7	Blanco
8	Gris
9	Azul claro
A	Verde claro
B	Cían claro
C	Rojo claro
D	Rosa claro
E	Marrón claro/amarillo
F	Blanco brillante

CARACTERES SUBRAYADOS

El modo 7 es el único que, de entrada, permite subrayar caracteres. En este modo, los atributos constan de 3 bits para primer plano, 3 bits para el fondo, y dos bits especiales, que controlan el brillo del primer plano y el parpadeo. Es posible modificar el comportamiento del bit 7 de los atributos, de modo que en lugar de controlar el parpadeo, se considere como bit de brillo del color de fondo. El conjunto de combinaciones posibles de valores de atributos está restringido. Vea en la Tabla 1 algunos ejemplos de Atributos en modo monocromo.

En los modos de 16 colores existen varios formatos que permiten distintas interpretaciones de los atributos. El primer formato, permite usar 4 bits para primer plano y 4 para el color de fondo. El segundo formato permite 4 bits para el color de fondo, 3 para el primer plano y 1 para seleccionar la fuente de caracteres (de las dos que pueden estar activadas simultáneamente). El tercer formato usa 3 bits para el fondo, 4 para el primer plano y 1 para activar el parpadeo. Por último, el cuarto formato es una combinación de los dos anteriores: se usa 1 bit de parpadeo, 1 bit de selección de fuente y 3 bits para fondo y primer plano, respectivamente. Consulte en la tabla 2 los colores que se corresponden con los valores cargados por defecto en los registros de la paleta de color, y que son los que se utilizan para interpretar los atributos de los caracteres.

CONCLUSIÓN

Para terminar, sólo queda comentar que existen numerosas funciones de la BIOS relacionadas con la gestión de los modos de vídeo alfanuméricos, y que, en algunos casos simplifican enormemente la tarea del programador. En el fichero TXTBIOS.TXT se muestran las funciones más importantes, y como ejemplo, se ha incluido un programa, que cambia la fuente de caracteres por defecto de la BIOS por una fuente definida por el usuario, de un aspecto un tanto desenfadado.

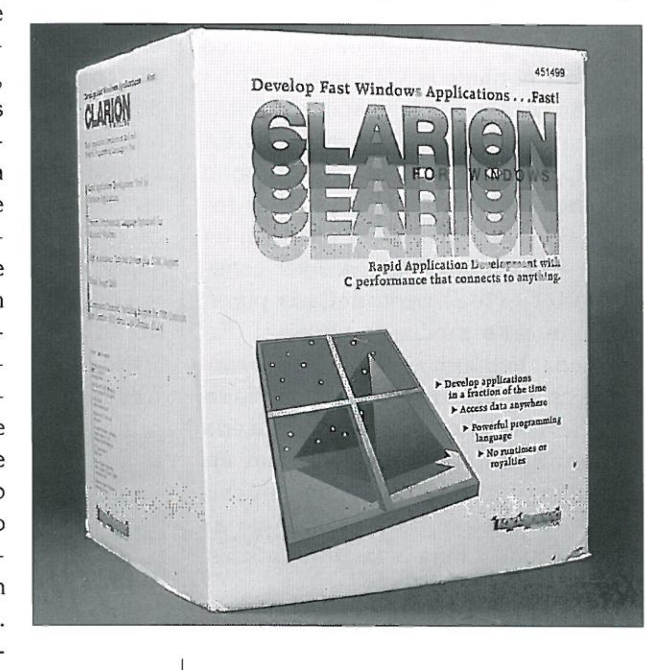
En el próximo capítulo de esta serie, nos ocuparemos de los diversos modos gráficos, y empezaremos a ver algunos efectos gráficos interesantes. ■

CLARION PARA WINDOWS

Tomás Tejón

La tendencia actual de los lenguajes de programación es la de ofrecer una serie de herramientas visuales que faciliten la labor del programador. Dentro de esta línea se encuadra Clarion, un lenguaje de programación que incorpora un generador de aplicaciones, mediante el cual es posible crear programas de bases de datos, sin escribir una sola línea de código.

El Lenguaje de programación Clarion no es algo nuevo, ya que sus orígenes se remontan al año 1986. La idea de su autor, Bruce D. Barrington, fue la de crear un lenguaje orientado hacia las tareas de empresa, y que resultase más sencillo de aprender que el resto de lenguajes de aquella época (BASIC, Cobol, Pascal...), pero sin perder la potencia de éstos. Tras un año de trabajo se llegó a la primera versión de Clarion, cuyas cualidades más destacadas eran las facilidades que incluía para el manejo de bases de datos, entrada por pantalla, e impresión de resultados. Esta versión funcionaba bajo MS-DOS, y utilizaba una técnica mixta de compilador-intérprete, mediante la cual el código fuente daba lugar a un código intermedio, que luego era interpretado en tiempo de ejecución, por un programa separado. Esta limitación desaparece en la versión del año 1990, que ya soportaba la compilación completa.



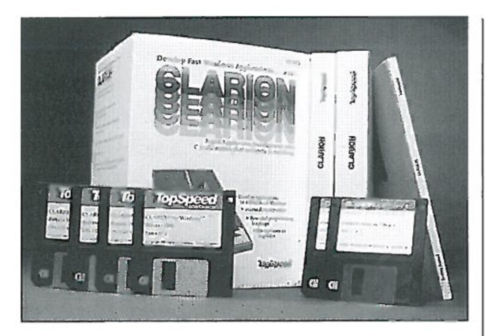
En el año 1992 la compañía JPI, con amplia experiencia en desarrollo de compiladores (Modula-2, C++, Pascal), se une a Clarion, formando así una nueva empresa bautizada como TopSpeed. El primer fruto de esta unión es el programa objeto de este artículo: La versión para Windows de Clarion.

EL DICCIONARIO DE DATOS

Lo más importante en una aplicación de base de datos son, evidentemente, los datos. Es por ello que, el primer paso para crear una aplicación de este tipo, es definir la estructura de los ficheros de datos que se necesitan, así como las relaciones que existen entre ellos.

En Clarion, ésto se gestiona mediante el diccionario de datos. El proceso que se sigue para cada fichero es el siguiente: en primer lugar se le asigna a la base de datos un nombre de fichero en el disco, así como su estructura física, es decir, el formato con que se guardan los datos. Se incluyen controladores para trabajar con datos en los formatos más conocidos de bases de datos: ASCII, BASIC, Btrieve, Clarion, Clipper, dBASE 3 y 4, DOS, FoxPro, Paradox, TopSpeed y cualquier otro sistema para el que exista un driver ODBC (Open DataBase Conectivity driver).

El siguiente paso es definir los campos que componen cada registro de la base de datos. Cada uno de estos



campos tiene asociado un tipo (cadena de caracteres, numérico, fecha...), una longitud y una serie de restricciones. Estas restricciones permiten limitar los valores que puede tomar el campo a unos valores determinados, obligar a que su valor coincida con algún otro o que no se pueda dejar su contenido en blanco. También se define aquí, el valor por defecto que toma cada campo y el tipo de control Windows que lo representará. Así, un campo que sólo puede tener, por ejemplo, cinco valores se puede representar en pantalla como una lista desplegable, o como una caja de botones.

Si la aplicación consta de varios archivos, seguramente existan relaciones entre ellos. Tal es el caso de un fichero de facturas, que se relaciona con otro en el que se encuentran almacenados los datos del cliente al que pertenece la factura. En este caso, deberá existir un campo idéntico en cada uno de los ficheros, que sirva para relacionarlos (en este caso sería un campo "código de cliente"). Existen tres tipos de relaciones posibles entre los registros del primer fichero con los del segundo, y que definen como propagar las modificaciones que se realicen en algún campo de la base de datos, para actualizar todos aquellos que están relacionados con él. Aparecen resumidas en la siguiente tabla:

Tipos de relaciones entre archivos de datos

Relación. A un elemento del primer fichero le corresponde: Cada elemento del segundo fichero puede pertenecer a: Ejemplo Notas

Uno a uno

(1:1) uno o ninguno del segundo uno del primero clientes<->num.FAX Representan datos opcionales

La ayuda incluye una referencia completa de todos los elementos del lenguaje con ejemplos

Uno a muchos

(1:n) uno o varios del segundo uno del primero clientes<->>facturas. Es el tipo de relación más frecuente

muchos a muchos

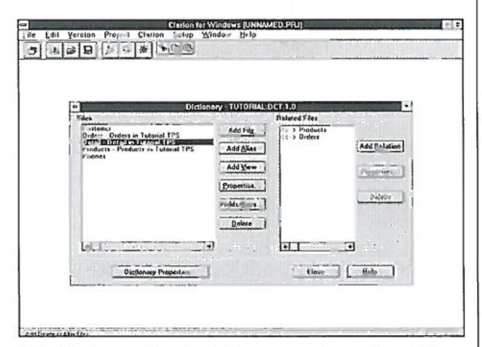
(n:n) uno o varios del segundo uno o varios del primero facturas<<->>productos Complejas y difíciles de mantener

Cada fichero tiene asociadas una serie de claves, a las que Clarion denomina keys, las cuales se forman mediante la combinación de uno o varios campos. Pueden ser de dos tipos: primarias, aquellas que identifican a un único registro y su valor no pueden repetirse, ni dejarse en blanco y externas, aquellas que se refieren a un campo que es clave en otro fichero de datos, y que relaciona a ambos archivos.

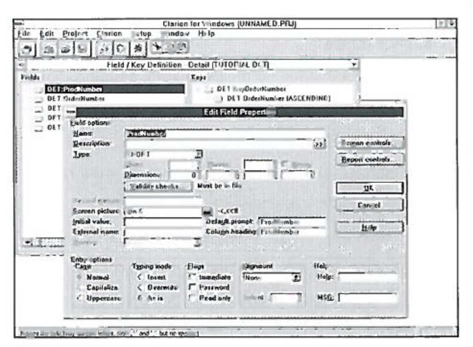
PLANTILLAS DE CÓDIGO

El sistema de desarrollo de aplicaciones de Clarion se ha hecho pensando en que resulte sencillo para la persona que lo utiliza, aunque no tenga grandes conocimientos del lenguaje. El sistema escogido consiste en una generación de código basada en templates, palabra inglesa que significa plantilla. El concepto de template es el de un esqueleto de programa, al que se le van añadiendo y configurando otros fragmentos de código (objetos), para ir formando así el programa definitivo. Cada uno de los objetos que se incluyen, se configura a través de unos cuadros de diálogo, definidos por el template, que contienen las opciones que los particularizan, para el uso concreto necesario en cada punto del programa. Aparte de este código preprogramado también es posible incorporar fragmentos de lenguaje Clarion, escritos por nosotros mismos, a este esqueleto. Para ello el template contempla unos puntos de inserción de código, cada uno de ellos con una misión muy específica, tales como inicializar algún campo, declarar variables o añadir funciones adicionales.

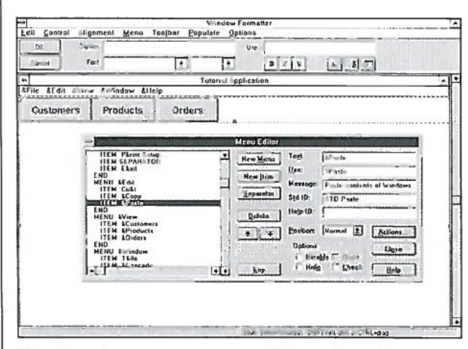
Clarion incorpora un template estándar como base del que, además, se incluye el código fuente. Es posible crear nuevas plantillas, o modificar la ya existentes, utilizando el lenguaje de



El diccionario de datos gestiona la información acerca de los ficheros de datos, los datos que contienen y las relaciones que existen entre ellos.

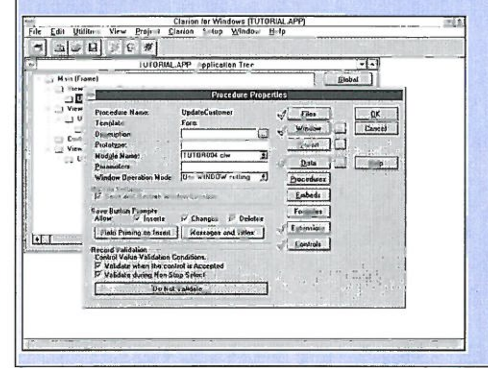


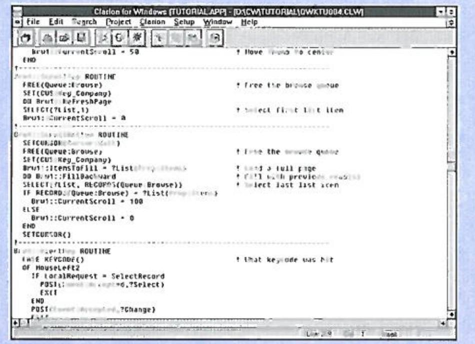
Cada uno de los campos de la base de datos tiene asociadas unas propiedades de formato en pantalla, formato en disco y valores que pueden tomar.



Clarion incorpora un editor visual y gestiona la parte del programa que se activará con cada una de las opciones del menú.

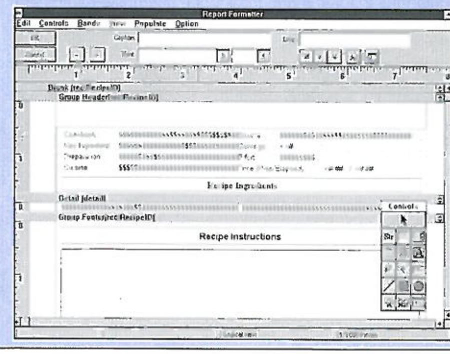
Cada uno de los objetos que se añaden al programa tiene una serie de características, que pueden ser editadas interactivamente mediante este programa.





El editor incorporado aumenta la legibilidad del código utilizando distintos colores para cada elemento sintáctico del lenguaje.

El diseñador de informes (reports) es una herramienta visual para definir el formato de los listados que generará nuestro programa.



template, que es una extensión del lenguaje de programación de Clarion diseñado para este fin.

EDITOR DE CÓDIGO

Para realizar modificaciones de código, el entorno integrado de Clarion incorpora su propio editor, especialmente diseñado para este fin. Entre sus características están el soporte de múltiples ficheros abiertos al mismo tiempo, búsqueda y sustitución de cadenas de texto, autoidentación y ayuda completa de todas las instrucciones del lenguaje, incluyendo ejemplos de cada una de ellas. Para aumentar la legibilidad del código, cada uno de los elementos sintácticos de Clarion aparece representado con un color distinto, algo que ya viene siendo habitual en los últimos compiladores del mercado. El editor tiene además otra utilidad, y es que a la hora de compilar el programa, si existen errores, estos aparecerán marcados en la ventana de edición, sobre la instrucción incorrecta.

EDITORES VISUALES

La escritura de código en Clarion ocupa un segundo lugar, quedando relegada a funciones de retoque o escritura de alguna que otra rutina. La mayor parte del trabajo se lleva a cabo a través del generador de aplicaciones, uniendo al cuerpo principal del programa objetos predefinidos. Los objetos que se incluyen aparecen en la siguiente tabla:

TIPOS DE OBJETOS EN CLARION

Browse. Muestra una serie de campos de uno o varios archivos que cumplen una determinada condición y permite desplazarse por ellos.Puede incorporar adicionalmente otros objetos, como botones para añadir, borrar o modificar registros y realizar búsquedas en la base de datos.

Código. Son fragmentos del lenguaje Clarion escritos por el desarrollador de la aplicación. Permiten programar acciones que no contempla el generador de código.

El sistema de desarrollo se ha hecho pensando en la persona que lo utiliza

Ventana. Una ventana es un contenedor de otros tipos de objetos. En ella se pueden incluir botones, menús, browses, campos de entrada. Se diseñan mediante un editor visual.

Menús. Mediante el editor de menús es posible generarlos de forma visual. A cada una de las opciones que se definan en el menú se le asignará el objeto que se activará, cuando el usuario escoja dicha opción.

Barra de Herramientas. Básicamente es lo mismo que un menú, tan únicamente varía su aspecto externo.

Campos de Entrada Permiten recoger un valor tecleado por el usuario, y asignárselo a un campo de la base de datos.

Botones. Un área rectangular que contiene un texto o dibujo. Ejecutan un comando o activan un objeto al ser pulsados.

RadioButtons. Es una lista de botones circulares, de los cuales sólo puede estar uno activo en un momento dado. Se corresponden con los valores que puede tomar un campo.

CheckButtons. Un recuadro que puede estar marcado, o no. Son equivalentes a un campo de tipo lógico, que puede tomar los valores cierto o falso.

El diseñador de informes posee una función de vista previa.

Controles VBX. Cualquier tipo de control creado para Visual Basic de Microsoft puede ser incluido en nuestras aplicaciones.

Report. Son listados por impresora. Se definen de forma muy similar a los browses, con la diferencia que contienen parámetros adicionales para es-

pecificar cabeceras, pies de página y control de la impresora.

Estos objetos se configuran mediante dos editores visuales: un editor de ventanas y un editor de listados.

El editor de ventanas toma como base una ventana vacía, y permite colocar sobre ella otros controles. Cada uno de estos controles se asociará con un campo (browse, checkbutton, campo de entrada) o con una llamada a otro objeto o rutina (código, botón, menú). Una vez completado el trabajo, el propio editor de ventanas creará el código necesario en nuestro proyecto.

El editor de listados es similar al anterior, con la diferencia que sólo admite dos tipos de controles: texto fijo y texto variable. El texto variable se asocia con algún campo de la base de datos o con alguna variable del programa (para numerar las páginas por ejemplo). Este editor incluye además una función de vista previa, que muestra el listado tal y como aparecerá en la impresora.

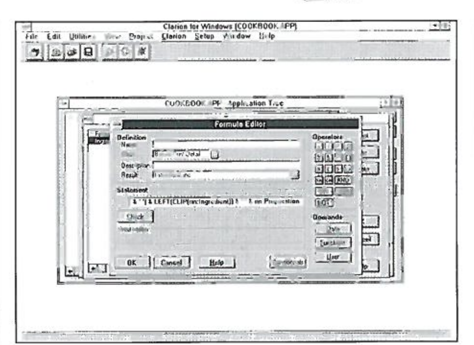
GESTOR DEL PROYECTO

El código generado por cada uno de los elementos que se añaden al programa se almacena en un archivo separado. El gestor de proyectos mantiene una lista de todos estos archivos, así como de librerías externas, otros ficheros con código del programador, controles VBX y ficheros objeto que necesita el programa. También se encarga de gestionar las opciones de compilación y optimización del código objeto. Es posible crear tanto ficheros ejecutables como librerías de tipo DLL para llamar desde programas escritos en otros lenguajes.

EL DEPURADOR

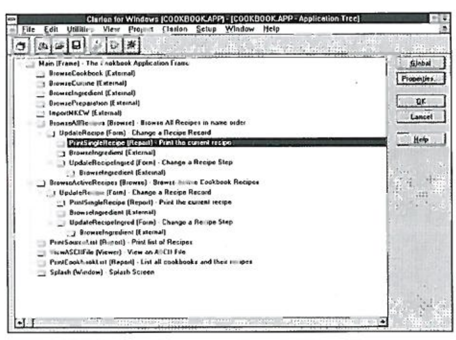
A la hora de buscar errores ocultos entre el código, el depurador incluido será de gran ayuda. Sin embargo esta herramienta funciona a nivel del código Clarion creado por el generador de aplicaciones, así que es necesario poseer un cierto nivel de programación del lenguaje para "entender" que sucede en cada momento. Por lo demás se trata de un depurador muy completo que admite puntos de ruptura condicionales, consultar el contenido de las variables en cualquier punto del programa. Se puede rastrear tanto el código fuente como el código maquina generado por el compilador.

Clarion soporta los formatos de archivo de los programas más extendidos de base de datos



A veces puede ser necesario introducir alguna fórmula para calcular el valor de un campo o verificar una entrada del usuario. El editor de fórmulas permite realizar este trabajo como si se manejase una calculadora.

Formatos de Bases de Datos soportados



Aplicación en desarrollo se muestra como una estructura jerárquica de los distintos objetos que la constituyen.

ASCII. Ficheros de texto sin delimitadores de campo. Los registros van separados por un salto de línea.

BASIC. Ficheros de texto con campos separados por comas. Los registros se separan por saltos de línea.

Btrieve. Sistema de bases de datos creado por Btrieve Technology Inc.

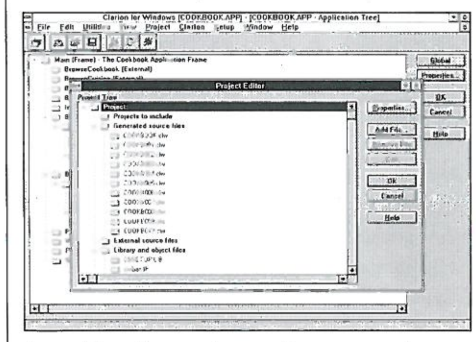
Clarion. Archivos creados por Clarion Database Developer 3.0 y Clarion Professional Developer.

Clipper. Soporta los formatos definidos por las versiones Summer '87 y Clipper 5.0.

dBASE. Existe soporte para los formatos de datos e índices que utilizan las versiones III y IV de este programa.

DOS. Utiliza ficheros con información binaria, sin delimitadores de campos ni de registros.

FoxPro. Este driver permite trabajar con archivos creados con FoxPro y FoxBase.



La aplicación en desarrollo se muestra como una estructura jerárquica.

Paradox. Compatible con todas las versiones de este programa anteriores a la 4.0.

TopSpeed. Sistema propio de Clarion para Windows. Es el más rápido de todos y soporta control de transacciones.

ODBC. Cualquier sistema de base de datos para el que exista un driver ODBC.

SQL-400. Disponible por separado. Trabaja con bases de datos de AS/400.

CONCLUSIÓN

La opción de generación de código es la baza fuerte de Clarion, pues permite realizar programas de gestión rápidamente y sin errores, incluso por personas con bajos conocimientos de programación. Por otra parte, al ser un lenguaje compilado, las aplicaciones generadas con él superan ampliamente en velocidad a las realizadas por otras herramientas similares. Sin duda, una opción a tener en cuenta. ■

TABLAS DESLIZANTES PARA MATRICES

Juan Manuel y Luis Martín

En el número 8 de esta revista, se explicó la forma de visualizar la información almacenada en bases de datos mediante el uso de tablas deslizantes. De esta forma, el contenido de cada registro se mostraba en las distintas filas de la tabla, mientras que cada columna mostraba los distintos valores de sus campos. Para ello, se utilizaban objetos de las clases TBrowse y TBColumn.

Sin embargo, estas dos clases también permiten crear tablas deslizantes para visualizar los elementos de una matriz. En este caso, la creación del objeto de la clase TBrowse no debe realizarse mediante el constructor TBrowseDB(), sino mediante el constructor genérico TBrowseNew(). La diferencia entre ambos radica en que, este último, no inicializa los valores de los atributos de movimiento, por lo que será necesario asignarlos explícitamente.

El modo de funcionamiento de las tablas deslizantes para matrices es muy similar al de las tablas para bases de datos. Sin embargo, la utilización de tablas deslizantes con matrices permite crear diferentes herramientas (menús de barras, sistemas de ayuda on-line, etc.).

Las tablas deslizantes para matrices permiten sustituir la función ACHOICE() por otras estructuras que facilitan al programador un mayor control sobre el proceso de selección. Además, a diferencia de dicha función, este tipo de tablas permite visualizar matrices bidimensionales.

El modo de funcionamiento de las tablas deslizantes para matrices es muy similar al de las tablas para bases de datos

Una de las herramientas más potentes que se ha introducido en el lenguaje Clipper, gracias a la incorporación a éste de la programación orientada a objetos, es la posibilidad de realizar tablas deslizantes. De esta forma, no sólo es posible visualizar la información contenida en bases de datos, sino también en otras estructuras de datos, como es el caso de las matrices.

CREACIÓN DE UNA TABLA DESLIZANTE

Como se a mencionado anteriormente, para crear un objeto de la clase TBrowse capaz de presentar una matriz en forma de tabla deslizante, debe utilizarse el constructor genérico de la clase, TBrowseNew(). El valor devuelto por éste método, debe ser asignado a una variable:

```
oTabla := TBrowseNew( nSup, nIzq, nInf, nDer )
```

En las tablas deslizantes para bases de datos, el puntero de la tabla deslizante coincide con el propio puntero de la base de datos. A diferencia de éstas, las tablas deslizantes para matrices necesitan utilizar una variable de memoria que realice esta función. Esta variable puntero o índice, contendrá la posición del elemento sobre el que se encuentra el cursor de la tabla. Por tanto, los bloques de código encargados del movimiento (GoToBlock, GoBottomBlock y SkipBlock) serán di-

CUADRO 1

```
#include "DBSTRUCT.CH"
#include "INKEY.CH"

FUNCTION Campos( cBase )

 LOCAL nIndice := 1, cColor

 SET( _SET_DATEFORMAT, "dd/mm/yyyy" )

 // Si no se pasa parámetro, visualizar sintaxis
 IF cBase == NIL
   ? "Sintaxis: CAMPOS <fichero.dbf>"; ?
   RETURN NIL
 ENDIF

 // Si no se encuentra el fichero
 IF !FILE( cBase )
   ? "No se encuentra el fichero " + cBase; ?
   RETURN NIL
 ENDIF

 // Cargar la estructura de la base de datos
 USE (cBase) NEW
 aDbf := DBSTRUCT()
 CLOSE ALL

 CLS
 cColor := SET( _SET_COLOR, "W/B+,N/BG" )

 // Crear el objeto tabla
 oTabla := TBrowseNew( 2, 22, 17, 55 )

 DISPBOX( 1, 21, 18, 56, 2 )

 // Añadir los objetos columna
 oTabla:AddColumn( TBColumnNew( "Nombre",;
   { || PADR( aDbf[ nIndice, DBS_NAME ], 10 ) } ) )
 oTabla:AddColumn( TBColumnNew( "Tipo",;
   { || PADR( aDbf[ nIndice, DBS_TYPE ], 1 ) } ) )
 oTabla:AddColumn( TBColumnNew( "Long.",;
   { || PADL( aDbf[ nIndice, DBS_LEN ], 3 ) } ) )
 oTabla:AddColumn( TBColumnNew( "Dec.",;
   { || PADL( aDbf[ nIndice, DBS_DEC ], 3 ) } ) )

 // Configurar el objeto tabla
 oTabla:ColSep := " 3 "
 oTabla:HeadSep := "ÄÂÄ"
 oTabla:GoTopBlock := { || nIndice := 1 }
  oTabla:GoBottomBlock := { || nIndice := LEN(
aDbf ) }
  oTabla:SkipBlock := { | n | n := Limita( LEN( aDbf
), nIndice, n ),;
    nIndice += n, n }

 // Bucle de control
 WHILE .T.
  // Estabilizar
  oTabla:ForceStable()
  // Procesar teclas
  nTecla := INKEY( 0 )
  DO CASE
   CASE nTecla == K_ESC      // Salir
    SET( _SET_COLOR, cColor )
    CLS
    QUIT
   CASE nTecla == K_UP
    oTabla:Up()
   CASE nTecla == K_DOWN
    oTabla:Down()
   CASE nTecla == K_LEFT
    oTabla:Left()
   CASE nTecla == K_RIGHT
    oTabla:Right()
   CASE nTecla == K_HOME
    oTabla:PanHome()
   CASE nTecla == K_END
    oTabla:PanEnd()
   CASE nTecla == K_PGUP
    oTabla:PageUp()
   CASE nTecla == K_PGDN
    oTabla:PageDown()
   CASE nTecla == K_CTRL_PGUP
    oTabla:GoTop()
   CASE nTecla == K_CTRL_PGDN
    oTabla:GoBottom()
  ENDCASE
 ENDDO

RETURN NIL


// Función para truncar los saltos
FUNCTION Limita( nUltimo, nActual, nSalto )

 IF nActual + nSalto < 1
   RETURN 1 - nActual
 ELSEIF nActual + nSalto > nUltimo
   RETURN nUltimo - nActual
 ENDIF

RETURN nSalto
```

Programa para visualizar la estructura de campos de una base de datos.

ferentes, y estarán referidos a dicho índice, según se verá más adelante.

La forma en que se crean y se añaden los distintos objetos columna es similar en el caso de bases de datos y de matrices. La única diferencia consiste en el bloque de código para la obtención de los valores. En el caso de una matriz, los valores devueltos por el bloque de código, deben estar referidos al índice de la misma. Además, se presenta un problema añadido. Cuando los valores de las columnas provienen de campos de una base de datos, su tipo y longitud son siempre los mismos. Sin embargo, los elementos de una matriz pueden ser de diferentes tipos de datos, y de distintas longitudes. Por tanto, es conveniente que los bloques de código que definen los valores de las columnas unifiquen, tanto el tipo, como el tamaño de los mismos. Para ello, la forma más sencilla es convertir los valores devueltos en cadenas de caracteres formateadas.

Si, por ejemplo, se desea visualizar en forma de tabla el contenido de la matriz estructural de una base de datos, puede utilizarse lo siguiente:

```
aDbf := DBSTRUCT()
nInd := 1
oTab := TBrowseNew( 5, 10, 19, 69 )

 oCol1 := TBColumnNew
            ( "Campo",; { || PADL
           ( aDbf [ nInd, DBS_NAME
                    ], 10 ) } )
 oCol2 := TBColumnNew
  ( "Tipo",;{ || aDbf[ nInd,
            DBS_TYPE ] } )
 oCol3 := TBColumnNew( "Long.",;
  { || PADR( aDbf[ nInd, DBS_LEN ],
            3 ) } )
 oCol4 := TBColumnNew( "Dec.",;
  { || PADR( aDbf[ nInd,
            DBS_NAME ] } )
```

Una vez creados todos los objetos columna, estos deben ser añadidos a la tabla deslizante:

```
oTab:AddColumn( oCol1 )
oTab:AddColumn( oCol2 )
oTab:AddColumn( oCol3 )
oTab:AddColumn( oCol4 )
```

LOS ATRIBUTOS DE MOVIMIENTO

Como se mencionó anteriormente, para la realización de tablas deslizantes de matrices es necesario utilizar el constructor genérico de la clase TBrowse. Ello trae consigo la necesidad de especificar los atributos de movimiento, es decir, los bloques de código encargados de situar el índice a lo largo de la matriz.

El atributo GoTopBlock debe contener un bloque de código que sitúe el

Para matrices es necesario utilizar una variable como puntero

CUADRO 2

```
#include "DIRECTRY.CH"
#include "INKEY.CH"

FUNCTION Fich()

 LOCAL aDir := {}, aMarca := {}
 LOCAL nIndice := 1, cColor
  LOCAL cCabecera := "  Fichero          Tamaño
Fecha  Hora   Atrib"
 SET( _SET_DATEFORMAT, "dd/mm/yyyy" )

 CLS
 cColor := SET( _SET_COLOR, "GR/R+,N/W" )

 // Cargar los ficheros del directorio actual
 aDir := DIRECTORY( "*.*", "HS" )

 // Inicializar la matriz de marcas
 AEVAL( aDir, { || AADD( aMarca, .F. ) } )

 // Crear el objeto tabla
 oDir := TBrowseNew( 2, 14, 17, 64 )

 DISPBOX( 1, 13, 18, 65, 2 )

 // Añadir el objeto columna
 oDir:AddColumn( TBColumnNew( cCabecera,;
  { || PADR( IF( aMarca[ nIndice ], "*", " " ), 2 ) +;
      PADR( aDir[ nIndice, F_NAME ], 13 ) +;
      PADL( aDir[ nIndice, F_SIZE ], 8 ) +;
      PADL( DTOC( aDir[ nIndice, F_DATE ] ), 11 )
+;
      PADL( aDir[ nIndice, F_TIME ], 9 ) +;
      PADL( LOWER( aDir[ nIndice, F_ATTR ] ), 4
) } ) )

 // Configuración de la tabla
 oDir:GoTopBlock := { || nIndice := 1 }
 oDir:GoBottomBlock := { || nIndice := LEN( aDir )
}
 oDir:SkipBlock := { | n | n := Limita( LEN( aDir ),
nIndice, n ),;
   nIndice += n, n }
 oDir:HeadSep := "Ä"

 // Bucle de control
 WHILE .T.
  oDir:ForceStable()
  nTecla := INKEY( 0 )
  DO CASE
   CASE nTecla == K_ESC
    // Salir al DOS
    SET( _SET_COLOR, cColor )
    CLS
    QUIT
   CASE nTecla == K_RETURN
    // Eliminar los ficheros marcados
    FOR i := 1 TO LEN( aDir )
     IF aMarca[ i ]
      ERASE (aDir[ i, F_NAME ])
     ENDIF
    NEXT
    SET( _SET_COLOR, cColor )
    CLS
    QUIT
   CASE nTecla == K_SPACE
    // Marcar el fichero actual
    aMarca[ nIndice ] := !aMarca[ nIndice ]
    oDir:RefreshCurrent()
   CASE nTecla == K_UP
    oDir:Up()
   CASE nTecla == K_DOWN
    oDir:Down()
   CASE nTecla == K_PGUP
    oDir:PageUp()
   CASE nTecla == K_PGDN
    oDir:PageDown()
   CASE nTecla == K_CTRL_PGUP
    oDir:GoTop()
   CASE nTecla == K_CTRL_PGDN
    oDir:GoBottom()
  ENDCASE
 ENDDO

RETURN NIL

// Función para truncar los saltos
FUNCTION Limita( nUltimo, nActual, nSalto )

 IF nActual + nSalto < 1
  RETURN 1 - nActual
 ELSEIF nActual + nSalto > nUltimo
  RETURN nUltimo - nActual
 ENDIF

RETURN nSalto
```

Programa que selecciona archivos para borrar.

puntero de la tabla en el primer registro o elemento. En el caso de bases de datos será { || DBGOTOP() }. En el caso de matrices, el bloque de código debe situar la variable de índice apuntando al primer elemento de la matriz, es decir, debe ser {|| nIndice := 1 }.

El atributo GoBottomBlock debe contener un bloque de código que sitúe el puntero de la tabla en el último registro o elemento. En el caso de bases de datos será {|| DBGOBOTTOM() }. En el caso de matrices, el bloque de código debe situar la variable de índice apuntando al último elemento de la matriz, es decir, debe ser {|| nIndice := LEN(aMatriz) }.

Por último, el atributo SkipBlock debe contener un bloque de código que sitúe el puntero en un determinado registro o elemento. Para ello, debe saltar desde el registro o elemento actual tantas posiciones como se le indique mediante un argumento numérico. Si este valor es positivo, el salto se realizará hacia el final, mientras que si es negativo, el salto se realizará hacia el principio. Debe tenerse en cuenta que un salto que supere las dimensiones, o el tamaño máximo debe ser truncado, tanto hacia el final como hacia el principio. Además, el bloque de código debe devolver un valor numérico correspondiente al número de posiciones que realmente se han saltado.

En el caso de bases de datos, el bloque será { | n | DBSKIP(n) }, ya que la función DBSKIP() realiza exactamente las tareas anteriormente descritas. Sin embargo, en el caso de matrices, el bloque de código es algo más complejo, y debe realizarse mediante la llamada a una función de usuario que realice los cálculos necesarios. El bloque de código puede ser el siguiente:

```
{ | n | n := Salto( LEN( aMatriz ),
                    nIndice, n ),;
  nIndice += n, n }
```

La función Salto() debe comprobar, que el número de posiciones que se desea saltar no exceda de los límites de la matriz. En dicho caso, el valor será truncado y devuelto.

Con matrices es necesario especificar los bloques de código de los atributos de movimiento

```
FUNCTION Salto
  ( nUltimo, nActual, nSalto )

  IF nActual + nSalto < 1
  // Si se sobrepasa el principio
  RETURN 1 - nActual
  ELSEIF nActual + nSalto > nUltimo

  // Si se sobrepasa el final
   RETURN nUltimo - nActual
  ENDIF

// Si el salto no sobrepasa los límites
RETURN nSalto
```



	Fichero	Tamaño	Fecha	Hora	Atrib
*	INSTALAR.EXE	434528	10/03/1992	12:00:00	a
	INSTALAR.HLP	63598	10/03/1992	12:00:00	a
	INSTALAR.TXT	42818	10/03/1992	12:00:00	a
*	SYSTEM.OLD	1772	16/02/1993	19:17:30	a
	WIN.INI	14821	15/03/1995	23:25:40	a
*	WINHELP.EXE	260080	10/03/1992	12:00:00	a
	CONTROL.HLP	137754	10/03/1992	12:00:00	a
*	WIN.COM	44916	28/03/1994	18:16:26	a
	BOOTLOG.TXT	1287	11/02/1993	19:14:56	a
	MOUSE.INI	28	11/02/1993	19:14:46	a
*	ACORDE.WAV	24982	10/03/1992	12:00:00	a
*	CALC.EXE	43488	10/03/1992	12:00:00	a
	CALENDAR.EXE	60608	10/03/1992	12:00:00	a
*	CANYON.MID	33883	10/03/1992	12:00:00	a

Figura 1. Tabla deslizante con ficheros marcados para borrar.

Una vez comprobado y rectificado el número de elementos a saltar, el bloque de código de movimiento debe sumar dicho valor al índice de la matriz. Este valor también debe ser devuelto por el bloque de código, por lo que será incluido como última expresión del mismo.

EL PROCESO DE VISUALIZACIÓN

El proceso de visualización de las tablas deslizantes para matrices es idéntico al de las tablas para bases de datos. Por tanto, el bucle de procesamiento de las teclas debe realizar primeramente la estabilización del objeto, ya sea ésta forzada o por etapas. Una vez que el objeto se encuentra estable puede llamarse a una función que se encargue de realizar las operaciones oportunas en función de la tecla pulsada. Para ello se hará uso de los distintos métodos que proporciona la clase TBrowse.

El proceso de visualización de las tablas deslizantes para matrices es idéntico al de las tablas para bases de datos

Para visualizar la tabla deslizante con la estructura de campos de una base de datos puede utilizarse el siguiente código:

```
WHILE .T.
  oTab:ForceStable()
  nTecla := INKEY( 0 )
  ProcesoTecla( nTecla )
ENDDO
```

Haciendo uso de los fragmentos de código utilizados, es posible realizar un sencillo ejemplo, que visualice la estructura de campos de una base de datos en forma de tabla deslizante. Para ello, bastará con cargar en una matriz, la estructura de la base de datos pasada como argumento desde la línea de comandos del DOS. El cuadro 1 muestra el listado completo de este ejemplo.

Como puede verse, cuando no se realizan operaciones sobre los datos de las columnas, no es necesario visualizar éstas como columnas independientes. En este caso basta con crear una sola columna que aglutine la información de todas las anteriores. Para ello, será necesario especificar un bloque de código que concatene los valores de las columnas. Para el ejemplo anterior, este bloque de código puede ser el siguiente:

```
oCol := TBColumnNew( cCabecera,;
  { || PADR( aDbf[ nInd, DBS_NAME ], 10 ) + " " +;
       aDbf[ nInd, DBS_TYPE ] + " " +;
       PADL( aDbf[ nIndice, DBS_LEN ], 3 ) + " " +;
       PADL( aDbf[ nIndice, DBS_DEC ], 3 ) } )
```

Un caso típico de utilización de una tabla deslizante con una única columna, consiste en la visualización de la matriz de ficheros de un directorio. En este tipo de matrices no es necesario operar sobre los distintos elementos del fichero (nombre, tamaño, atributos, etc.). El cuadro 2 muestra el listado de un programa que permite seleccionar aleatoriamente los ficheros del directorio actual que se desea borrar. Para ello, se visualiza una tabla deslizante de una única columna con la información de cada fichero, como puede verse en la figura 1.

Al pulsar la tecla Espaciador, se van marcando o desmarcando los ficheros a borrar. El borrado de todos los archivos marcados se realiza pulsando la tecla Intro. Si se pulsa la tecla Escape se aborta la operación. Para realizar el proceso de marcación se utiliza una matriz paralela con valores lógicos, de forma que un valor .T. indica que el fichero va a ser borrado.

CONCLUSIÓN

Con el presente artículo se pretende fomentar la utilización de los objetos de la clase TBrowse para realización de tablas y menús. Aunque la utilización de la función ACHOICE() es más sencilla, sus limitaciones no permiten realizar aplicaciones de mayor complejidad. Por tanto, es recomendable la utilización de tablas deslizantes basadas en objetos, ya que proporcionan un mayor control y versatilidad. ■

NETBIOS (3ª PARTE)

Carlos Arias

En los capítulos anteriores se vio el manejo del controlador NETBIOS y sus comandos y buffers de datos asociados. También se comentó la función 5Ch de MSDOS, utilizada para comunicarnos con el controlador.

Aunque el conocimiento de toda esta información es indispensable para poder desarrollar con éxito una aplicación de red basada en el interfaz NETBIOS, no es suficiente, puesto que antes de poder ejecutar comunicación alguna, hay que realizar ciertas averiguaciones que atañen al sistema de red que se esté utilizando.

COMPROBACIONES PRELIMINARES

Antes de poder llamar a NETBIOS hay que asegurarse que éste está instalado en memoria.

Para este fin, MSDOS cuenta a partir de la versión 3.1 con una serie de funciones para facilitar la programación en entornos de red. Algunas de estas funciones no están documentadas por Microsoft, pero su utilización en numerosas aplicaciones aseguran su compatibilidad con futuras versiones del DOS. Las funciones son la 5Eh, 5Fh y la interrupción indocumentada 2Ah. Las dos primeras se dividen a su vez en las subfunciones 5E00h, 5E02h, 5E03h, 5F02h, 5F03h y 5F04h. Todas ellas, excepto la indocumentada 2Ah, se llaman a través de la interrupción de servicios del DOS 21h.

La subfunción 5E00h devuelve el nombre de la estación de trabajo si el ordenador está conectado a una red. Si tiene éxito devuelve el carry flag a cero, y el contenido del registro CH a cero si el nombre no está definido, o distinto de cero si ya está definido. Además, devuelve en el registro CL el número de versión BIOS de red, si CH es distinto de cero. El par de registros DS:DX contienen la dirección de segmento y desplazamiento del identificador de la estación de trabajo.

Si la llamada fracasa, se devuelve el código de error extendido en el contenido del registro AX, código que se puede utilizar para llamar a la función de servicio de error extendido para obtener una explicación.

Antes de llamar a esta subfunción, hay que cargar el comando SHARE en memoria. Si no se encuentra instalado, la llamada puede tener resultados impredecibles.

DETECCIÓN DE LA PRESENCIA DE SHARE

Para poder detectar si SHARE se encuentra instalado en memoria, hay que realizar una llamada a una función del multiplexor.

El multiplexor del DOS no es más que un servicio de interrupción del DOS pensado para ser utilizado por los programas TSR que se instalan en el ordenador y necesitan de una interfaz de comunicación con el exterior. Tal es el caso de los comandos del DOS Keyb, DosKey, Share, Print, Assign, ANSY.SYS, etc.

Todos los programas que deseen utilizar los servicios del multiplexor, tienen que asignarse un número identificativo de 8 bits. Microsoft tiene reservados para sus programas TSR los números entre el 00h y el BFh, dejando del C0h al FEh para utilización de terceras partes. Para comprobar si un programa de este tipo se encuentra instalado en memoria o para instalarlo por primera vez, se llama a la interrupción del multiplexor 27h, enviando como parámetros el número identificativo del programa en el registro AH, y el valor cero en el registro AL. Si al retornar de la interrupción el registro AL contiene un valor distinto de cero, normalmente el FFh, significa que el programa está ya instalado en memoria.

En el caso particular de SHARE, se llama al multiplexor con el valor 10h,

1º Comprobar que está cargado SHARE.
2º Comprobación de existencia de entorno de red mediante interrupción indocumentada 2Ah.
3º Obtener nombre del ordenador en la red mediante la subfunción 00h de la función 5Eh.
4º Obtener la tabla de redirecciones mediante el uso repetido de la subfunción 02h de la función 5Fh.

CUADRO 1: Pasos a seguir para la utilización de las funciones relacionadas con la redirección de dispositivos y gestión de impresoras.

que es el número identificativo de SHARE, en el registro AH, y 00h en AL. Si al retornar de la interrupción AL contiene el valor FFh, significa que está instalado. Si no está instalado, habrá que finalizar la aplicación de red, advirtiendo al usuario de la necesidad de instalarlo.

INTERRUPCIÓN INDOCUMENTADA 2AH

Debido a que los resultados obtenidos de la llamada a la subfunción 5F00h pueden pertenecer a un dispositivo de red desinstalado, hay que utilizar la interrupción indocumentada 2Ah, pasándola como único parámetro el valor 0h cargado en el registro AX. Ya que esta interrupción lo único que ejecuta es una instrucción IRET en caso de no estar instalado un controlador de red, se puede afirmar que si a la salida no ha modificado el contenido del registro AX, no hay ninguna red instalada. En caso contrario, sí existe una red. Es por tanto la manera más fiable de detectar la presencia de una red.

RECURSOS COMPARTIDOS

Si en la aplicación se van a utilizar recursos compartidos, las subfunciones 02h, 03h y 04h de la función 5Fh dan información y permiten la modificación de la lista de los recursos asignados al ordenador. Estas funciones son aplicables sólo a la red Microsoft Network o compatibles. La lista de asignaciones es una tabla que contiene información de los dispositivos de disco, e impresoras pertenecientes a servidores u otras estaciones de trabajo, que la estación está utilizando como si fuesen suyas. En otras palabras, da información de los dispositivos redireccionados. Las unidades de disco, ficheros y directorios compartidos tienen el valor 04h. Las impresoras y unidades de fax tienen asignados el valor 03h.

La subfunción 5F02h devuelve información de un elemento de la lista de asignaciones. Cada dispositivo de la lista está referenciado por un número de índice. De esta forma se le indica a la función el índice del dispositivo de la lista del que se desea obtener la información. Se la pasan como parámetros la siguiente información:

- el registro AX debe contener el valor 5F02h.
- el número de índice en la lista de dispositivos hay que pasarlo en el registro BX. Si se le pasa un número de índice mayor al número total de elementos de la lista, devuelve el carry flag a 1 y el código de error 18 en el registro AX. Un método para saber el número total de dispositivos consiste en asignar al registro BX un valor de cero e ir incrementando dicho valor en sucesivas llamadas a la función hasta que el carry flag tome valor 1 y el registro AX valga 18.
- DS:DX debe de contener la dirección de memoria donde se quiere que la función describa mediante una cadena ASCIIZ (cadena ASCII terminada con un carácter nulo) el nombre asignado por la estación de trabajo al dispositivo local.
- ES:DI apunta a la dirección de memoria donde la función describirá mediante una cadena ASCIIZ el nombre del ordenador remoto que contiene el recurso, y el nombre lógico por el cual se conoce al dispositivo dentro de la red.

Los valores devueltos son los siguientes:

- si la llamada tuvo éxito el carry flag devuelve cero.
- el registro AX contiene el código de error si la llamada no tiene éxito. Un valor de 18 indica que se ha pasado a la función en el registro BX un número índice mayor que el número total de dispositivos redireccionados.
- CX contiene los parámetros del dispositivo. Estos se establecen mediante la función 5F03h.
- BH vale cero si el dispositivo es válido. En caso contrario vale 1.

La función 5F03h permite redirigir un dispositivo disponible en la red hacia una unidad de dispositivo local. Por ejemplo, se puede asignar como unidad lógica E: una unidad de disco duro de la red que pertenezca a un servidor u otra estación de trabajo. Los parámetros de entrada son los siguientes:

Subfunción 00h:
Obtiene el nombre dado al ordenador en la red.
Parámetros de entrada:
DS:DX Dirección donde se recibe el nombre del ordenador como una cadena ASCIIZ.

Valores de salida:
Indicador de acarreo bajo si el ordenador está registrado en la red.
CH Es 0 si el nombre no está definido.
CL Número de versión de la red.
AX Código de error si la llamada falló.

Subfunción 02h
Añade una cadena de inicialización a la impresora de red.
Parámetros de entrada:
BX Número índice que tiene asignado la impresora en la tabla de redireccionamiento.
CX Longitud de la cadena de inicialización.
DS:DX Cadena ASCII de inicialización.

Valores de salida:
Indicador de acarreo bajo si el ordenador está registrado en la red.
AX Código de error si no se ejecutó correctamente.

Subfunción 03h
Obtiene la cadena de inicialización de una impresora de red. Es la complementaria de la anterior.
Parámetros de entrada:
BX Número índice que tiene asignado la impresora en la tabla de redireccionamiento.
ES:DI Dirección donde se devolverá la cadena de inicialización.

Valores de salida:
Indicador de acarreo bajo si el ordenador está registrado en la red.
AX Código de error si no se ejecutó correctamente.

CUADRO 2: Subfunciones de la función 5Eh.

Subfunción 02h:
Obtiene un elemento de la lista de redirecciones.
Parámetros de entrada:

BX	Número índice del dispositivo en la lista de redirecciones.
DS:DX	Dirección donde la subfunción escribirá como cadena ASCIIZ el nombre del dispositivo local asociado al número índice. Por ejemplo, LPT1, LPT2, E:, PRN, etc.
ES:DI	Dirección donde la subfunción escribirá como cadenas ASCIIZ el nombre del ordenador remoto que es propietario del dispositivo, el nombre del dispositivo dado por dicho ordenador y la palabra clave de acceso.
AX	Código de error si el indicador de acarreo está a 1.

Valores de salida:
Indicador de acarreo bajo si el ordenador está registrado en la red.

BH	El bit 0 indica si el dispositivo es válido o no valido. Si es cero indica dispositivo válido.
BL	Si vale 3 es un dispositivo de impresora. Si vale 4 es una unidad de disco, fichero o directorio.
CX	Parámetros dados al dispositivo mediante la subfunción 03. El significado de ellos los fija el programador de la aplicación.
DS:SI	Mismo significado que el parámetro de entrada pasado en DS:DX.
ES:DI	Mismo significado que el parámetro de entrada ES:DI.

Subfunción 03h:
Redirecciona un dispositivo.

BL	Indica el tipo de dispositivo a redireccionar. 3 para dispositivo de impresora y 4 para una unidad de disco, fichero o directorio.
CX	Parámetros dados al dispositivo. Su significado los fija el programador de la aplicación.
DS:SI	Dirección donde la subfunción escribirá como cadena ASCIIZ el nombre del dispositivo local asociado al número índice. Por ejemplo, LPT1, LPT2, E:, PRN, etc.
ES:DI	Dirección donde la subfunción escribirá como cadenas ASCIIZ el nombre del ordenador remoto que es propietario del dispositivo, el nombre del dispositivo dado por dicho ordenador y la palabra clave de acceso.

Valores de salida:
Indicador de acarreo bajo si se ejecutó con éxito.

AX	Código de error.

Subfunción 04h:
Cancela la redirección de un dispositivo.
Parámetros de entrada:

DS:SI	dirección de la cadena ASCIIZ con el nombre del dispositivo local a cancelar.

Valores de salida:
Indicador de acarreo bajo si se ejecutó con éxito.

AX	Código de error

CUADRO 3: Subfunciones de la función 5Fh. Están relacionadas con la gestión de los dispositivos redireccionados. Como en el cuadro anterior, se pasa el registro AH con el valor 5Fh y el registro AL con el número de subfunción a llamar.

- AX contiene el valor 5F03h.

- BL debe de contener el tipo de dispositivo. El valor 3 significa impresora y el valor 4 indica una unidad de disco.

- CX contiene los parámetros que se le asignan al dispositivo. Estos parámetros los asigna el programador que desarrolla la aplicación.

- DS:SI apunta a una cadena ASCIIZ con el nombre del dispositivo local. Por ejemplo, si se va a asignar una unidad de disco duro a la unidad lógica E, contendrá "e:\0"

- ES:DI apunta a dos cadenas ASCIIZ contiguas. La primera se refiere al nombre del ordenador que tiene el dispositivo seguido del nombre dado en la red al dispositivo. La segunda cadena indica la palabra clave de acceso. Por ejemplo, para referirse al directorio REDES del servidor SOLOPROGRAMADOR utilizando la clave PAISE, la cadena contendrá el texto "\SOLOPROGRAMADOR\REDES",0,"PAISE",0.

Al retornar de la llamada a la función, se obtiene el siguiente resultado en el registro AX y en el carry flag:

- carry flag a cero si se ejecutó con éxito la función.

- AX contiene el código de error si el carry flag está a 1.

La función 5F04h cancela la redirección de un dispositivo liberando la unidad de dispositivo lógica asignada a una unidad de dispositivo asignada en la red. Los parámetros de entrada son los siguientes:

- AX con el valor 5F04h.

- DS:SI apunta a una cadena ASCIIZ con el nombre del dispositivo local a liberar.

Si la función tuvo éxito el carry flag lo devuelve bajo. En caso contrario, el registro AX contiene el código de error.

CONTROL DE LA IMPRESORA

El MSDOS nos permite mediante las subfunciones 02h y 03h de la función 5Eh añadir una cadena de inicialización a la impresora de red, y de obtener los parámetros de la impresora.

La subfunción 02h de 5Eh sirve para añadir una cadena de control a los trabajos enviados a una impresora de la red, de tal manera que se pueda personalizar el funcionamiento para cada puesto de trabajo. Para indicar la impresora de red a la que se quiere añadir la cadena de caracteres, se pasa a la función el número índice que tiene asignado la impresora en la tabla de redirecciones. Los parámetros que se dan a la función son los siguientes:

- AX con el valor 5E02h.

- BX con el número índice que tiene asignado la impresora en la lista de redirecciones.

- la longitud de la cadena de inicialización se pasa en BX, y la dirección donde se encuentra se haya en DS:SI.

El resultado de la función se obtiene en el carry flag y en el registro AX, conteniendo este último el código de error en caso de no haber ejecutado con éxito la llamada.

Por último, se dispone de la función inversa de la anterior. Con la subfunción 03h de 5Eh se obtiene la cadena de inicialización de una impresora de red. Al igual que en la función anterior, se pasa el número índice que tiene asignado la impresora en la tabla de redirecciones. Los parámetros de entrada son los siguientes:

- AX con el valor 5E03h.

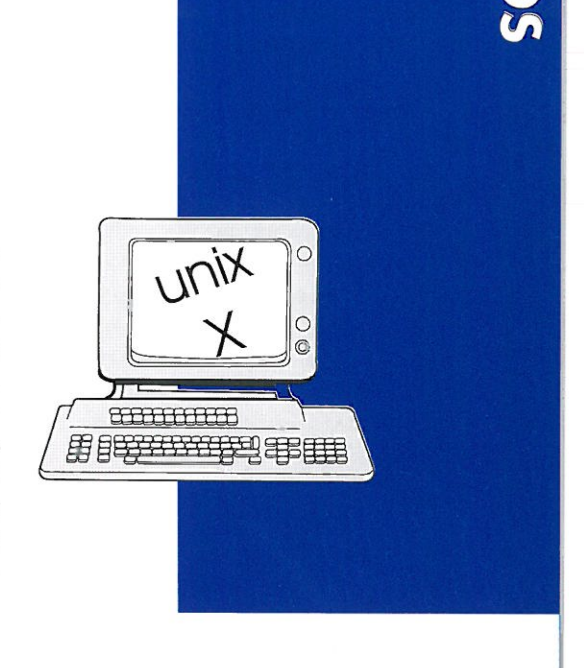
- BX contiene el número índice dado a la impresora en la tabla de redirecciones.

- ES:DI contiene la dirección de memoria donde la función escribirá la cadena de inicialización.

Si la función tiene éxito, además de devolver el resultado en AX y en el carry flag, CX contiene la longitud de la cadena de inicialización. ■

LOS LENGUAJES DE PROGRAMACIÓN

Fernando de la Villa



El sistema operativo Linux se está convirtiendo en una interesante plataforma para la creación de programas. Día tras día crece el número de usuarios, que se sienten atraídos por este sistema, lo que conlleva la aparición de nuevos programadores que escriben software para Linux. La oferta es muy amplia: desde programas de comunicaciones hasta emuladores de DOS y Windows, pasando por juegos tan populares como el DOOM.

Lo cierto es que Linux está muy bien provisto de herramientas para el desarrollo de programas. Se dispone de diversas librerías, utilidades, compiladores y, en la mayoría de los casos, una buena documentación, que facilita la tarea del sufrido programador.

Linux dispone actualmente de varios compiladores e intérpretes de lenguajes como Ada, Prolog y Fortran. Sin embargo, en este artículo se tratan los dos lenguajes incluidos en Linux Slackware 2.1.0.

EL COMPILADOR GNU-C

En realidad no se trata de un compilador, si no de tres reunidos en uno sólo. GCC (GNU C Compiler), es capaz de tratar con código fuente en C, en C++ y en Objetive-C, lo que le convierte en un compilador muy versátil. Gran parte de su potencia reside, en que está diseñado de forma independiente a la máquina, por tanto puede ser utilizado como compilador cruzado entre diferentes sistemas. Es decir, con GCC se puede compilar en Linux un programa destinado a correr en BSD, SCO o SunOS. Pese a su generalidad, proporciona gran cantidad de opciones para la optimización del código generado, algunas de ellas específicas del procesador incorporado en las diferentes máquinas, y además, opciones para el enlace, ensamblado, depuración y mensajes de aviso. En total, se dispone de más de trescientas opciones diferentes.

INSTALACIÓN

Slackware Linux 2.1.0 incorpora la versión 2.5.8 de GCC. Es importante conocer este dato, para compilar los programas que necesitan una determinada versión (o superior) del compilador. Para obtener esta información se dispone de la opción de llamada —version. Con Slackware se puede instalar GCC de varias formas que se verán a continuación. Para explicar el proceso se asume que la unidad de CD-ROM está montada en el directorio /cdrom.

El primer método es, durante la instalación de Linux, en el momento de seleccionar lo que se quiere instalar. Bastaría con señalar entre los paquetes elegidos, el grupo D (Program Development). LLegado el momento, se hacen unas preguntas sobre, qué se desea instalar, del grupo de paquetes de desarrollo. Para poder compilar, como mínimo se deben instalar los paquetes GCC, binutils, include, libc, y la tercera parte de las fuentes del kernel

El número de programadores que se deciden por Linux está creciendo

Dado el gran interés de los lectores por el sistema operativo Linux, el capítulo de Sistemas Abiertos está dedicado este mes a las herramientas de programación (compiladores, montadores, etc), incluidas en el CD-ROM que se regaló en el número 6 de Sólo Programadores.

o núcleo del sistema operativo. En el supuesto de que se necesite compilar en C++, además es necesario instalar gxx y libgxx.

El segundo es, una vez instalado el sistema operativo, mediante el uso de la utilidad setup. En primer lugar se seleccionará la opción "source", para indicar desde dónde se va a instalar el paquete. A continuación se elige la opción "Install from a pre-mounted directory", y a la pregunta sobre el directorio en el que se encuentra la distribución, se responde con /cdrom/linux210. Posteriormente se marca el paquete D, y se continua el proceso de instalación, siguiendo las instrucciones de la pantalla. Posteriormente se instalarán los paquetes indicados en el párrafo anterior, para poder compilar sin problemas.

El tercer método es el más complicado de todos. Mediante la utilidad pkgtool es factible la instalación de paquetes desde el directorio en que estén situados en la kit de distribución. El mayor problema es, que no permite la instalación de un grupo de paquetes, sino de paquetes situados en un sólo directorio. Por esta razón, no hay más remedio que llamar varias veces a pkgtool para instalar un grupo de paquetes. El grupo de desarrollo de programas viene en 7 directorios: desde el d1 hasta el d7. Para instalar lo mínimo necesario, se necesitan los directorios d1, d2, d4, d5 y d7.

En primer lugar hay que indicar en el menú de la utilidad la opción "other", para instalar desde un directorio distinto del actual. Después es preciso teclear el nombre del directorio, que será /cdrom/linux210/dnum, donde num será un número del 1 al 7. Y como siempre, instalar los paquetes citados en el primer método.

UTILIZACIÓN

Una vez instalado, podemos comprobar el correcto funcionamiento del compilador mediante un pequeño programa de prueba. Por ejemplo, el consabido "¡Hola, mundo!", que se inventó para este tipo de cosas:

```
#include <stdio.h>

int main () {
  printf ("¡Hola, mundo!\n");
  return 0;
}
```

Para poder compilar, se teclea el programa con un editor de texto sin formato como el vi, se graba con el nombre hola.c y se introduce la orden:

```
cc hola.c
```

Si todo ha ido bien, no saldrá ningún mensaje y se habrá creado en el directorio actual un fichero llamado a.out. Éste es el nombre por defecto del ejecutable resultante del proceso de compilación y enlace. Podemos cambiar el nombre, utilizando la opción -o:

```
cc -o hola hola.c
```

GCC es un compilador "inteligente". Esto quiere decir que según el sufijo de los ficheros de entrada, actúa en consecuencia. En la tabla 1 pueden verse los diferentes sufijos reconocidos por el compilador, y las acciones que realiza el mismo con cada fichero según su sufijo.

El orden normal de actuación del compilador es preprocesado, compilado, ensamblado y enlace. Sin embargo, el proceso puede ser finalizado antes de entrar en cualquiera de las tres últimas etapas. La opción -c indica no enlazar, -S no ensamblar y -E no compilar. Esto puede ser útil para ver el resultado del preprocesamiento, para depurar a mano el código en lenguaje ensamblador generado por el compilador, o para crear librerías con funciones que se hayan escrito anteriormente. Por ejemplo, para crear el código objeto de tres funciones en C, contenidas en los archivos fich1.c, fich2.c y fich3.c se teclearía:

```
cc -c fich1.c fich2.c fich3.c
```

Los archivos fich1.o, fich2.o y fich3.o obtenidos podrían ser añadidos a una librería ya existente, o crear una librería nueva mediante la orden ar. Esta orden sirve a los programadores para realizar el mantenimiento de librerías de enlace estático (enlace en tiempo de compilación). Se puede crear una librería con los tres ficheros citados con la orden:

```
ar vr libmia.a fich1.o fich2.o fich3.o
```

Por otro lado, pueden hacerse combinaciones con los distintos tipos de fichero en la llamada. Como se mencionó anteriormente, GCC es un compilador inteligente. Para entender las acciones que realiza el compilador, lo mejor es un ejemplo:

```
cc -o prueba uno.c dos.s
```

GCC ante esta entrada, preprocesa, compila y ensambla el archivo uno.c, ensambla el archivo dos.s y por último (si no hubo errores) enlaza los objetos resultantes del proceso anterior en el ejecutable prueba. La gran ventaja

. C	Fuente en C	Procesar, compilar y ensamblar
. C .cc .cxx	Fuente en C + +	Procesar, compilar y ensamblar
. m	Fuente en Objetive-C	Procesar, compilar y ensamblar
. i	C preprocesador	Compilar y ensamblar
. s	Fuente en ensamblador	Ensamblar
. S	Fuente en ensamblador	Preprocesar y ensamblar
. o	Fichero objeto	
. a	Fichero de archivo	
. h	Fichero de inclusión	

Sufijos reconocidos por GCC y acciones relacionadas.

que proporciona esta característica de GCC, es que se evita el tener que recompilar todos los ficheros relacionados con un programa cada vez que se hace una modificación en uno de los archivos.

Si el programa se compone de un gran número de ficheros, es conveniente el uso de la utilidad make. Sin entrar en detalles, dicha orden sirve para detectar las partes de un programa que han sido modificadas, y ejecuta los comandos pertinentes para recompilarlas.

Si lo que se persigue es la escritura de programas C++, hay que utilizar la orden g++ en lugar de cc. En realidad, sólo hay un compilador, pero g++ es la manera de usar GCC con las opciones necesarias para compilar código fuente en C++. Con el fin de tener un mayor control sobre la compilación, existen opciones específicas de cada lenguaje, como -fno-strict-prototype para C++ y -ansi para C.

Linux proporciona un gran número de herramientas para la programación

Aparte de las utilidades mencionadas, existen otras relacionadas con GCC. El preprocesador de C (cpp), que aunque es llamado automáticamente por el compilador, puede ser útil en algunas ocasiones. Otra herramienta muy empleada es el GNU-Debugger (gdb), que permite la depuración de código C y C++, y que dispone de un interfaz para X-Window denominado xxgdb.

Para más información, consulte las páginas del manual de Linux (man), y la información accesible mediante el visualizador de hipertexto info.

EL INTÉRPRETE GNU-SMALLTALK

A diferencia de GCC, GNU-Smalltalk es un intérprete, no un compilador. Esto significa que no es capaz de generar código, por lo que siempre se necesita el intérprete para poder ejecutar programas y, como es lógico, la ejecución es más lenta. La versión 1.1.1 de GNU-Smalltalk es una implementación bastante fiel para máquinas UNIX de la especificación del lenguaje orientado a objetos Smalltalk-80. Por desgracia, no incluye algunos detalles del lenguaje como enteros largos y señales asíncronas. A cambio incorpora varias mejoras al lenguaje, específicas de este intérprete.

INSTALACIÓN

La instalación del paquete Smalltalk es muy parecida a la descrita para el compilador GCC. Los archivos necesarios se encuentran en el directorio /cdrom/linux210/oop1. Cualquiera de los tres métodos especificados anteriormente es válido para llevar a cabo la instalación.

UTILIZACIÓN

La primera vez que se llama al intérprete (mst), se observa la aparición de un mensaje del tipo "GC flipping to space 1...copied space = 100.0%...done". Esto sucede porque GNU-Smalltalk utiliza para su funcionamiento lo que denomina una "máquina virtual". Esto no es otra cosa que la grabación del estado del intérprete en un archivo, para reanudar el proceso más tarde de una manera rápida. La primera vez, se cargan una serie de ficheros necesarios para crear el kernel de Smalltalk, se realiza un proceso de limpieza llamado "recolección de basura", y se graba la tabla de objetos y la memoria que estos utilizan en el fichero mst.im. Cada vez que entra en funcionamiento mst, lo primero que hace es buscar en el directorio actual dicho fichero, que se correspondería a la imagen de la máquina virtual Smalltalk del propio usuario. Si no lo encuentra, pasa a buscar el mst.im del sistema, situado en el directorio /usr/lib/smalltalk. Si no existe, se crea en el directorio actual. En todo caso, el fichero se vuelve a crear si se han producido modificaciones o si se fuerza su creación con la opción -i del intérprete.

La aparición del siguiente mensaje en pantalla es el indicador de que todo funciona:

```
Smalltalk 1.1.1 Ready

st>
```

La segunda línea es el prompt, esto es, la invitación a teclear el código del programa. Para probar el intérprete se puede intentar que Smalltalk diga (cómo no) ¡Hola, mundo!. Será necesario crear un objeto de tipo cadena y enviarle un mensaje ordenándole que se imprima en pantalla:

```
'¡Hola, mundo!' printNl !
```

El signo de exclamación final indica a GNU-Smalltalk que puede comenzar a procesar la información. Aparecerá el mensaje: "¡Hola, mundo!", junto con una serie de datos de tipo estadístico sin demasiado interés, excepto el del tiempo de ejecución del programa. La presentación de estos datos puede ser evitada con la inclusión de la opción -q en la llamada.

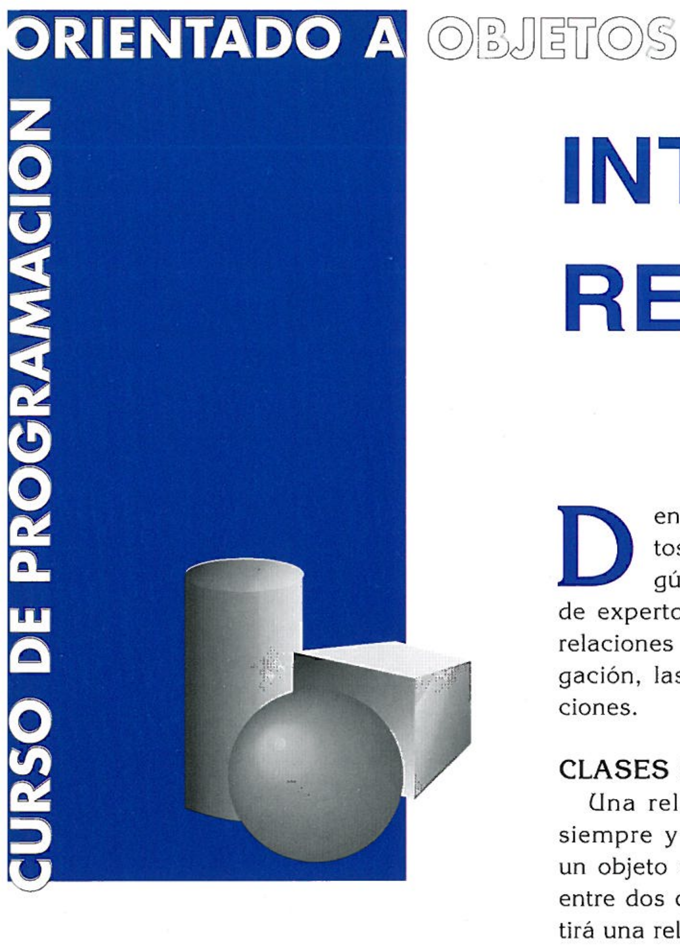
Otra forma de ejecutar programas es, pasando el nombre de un fichero o ficheros que contengan código fuente Smalltalk en la línea de comandos, siendo el orden de ejecución de los programas exactamente el indicado en la llamada:

```
mst -q primero.st segundo.st
```

Algunas opciones útiles son -d para seguimiento de declaraciones, -I <fichero> para cargar un archivo imagen diferente al mst.im y -h para ver la ayuda del intérprete.

GNU-Smalltalk viene acompañado de stix (SmallTalk Interface to X), un pequeño interfaz del lenguaje para X-Window. Sin embargo, este interfaz es más bien experimental, al no poseer una implementación completa del protocolo X. Los ficheros relacionados con el mismo están situados en el directorio /usr/lib/stix.

Se puede encontrar más información en el visualizador info y en los documentos situados en /usr/lib/smalltalk/documentation y /usr/lib/stix. ■



INTEGRIDAD RELACIONAL

Ignacio Cea

Dentro de la orientación a objetos es necesario destacar, según criterio de la mayor parte de expertos en el área, tres tipos de relaciones entre objetos: las de agregación, las de herencia y las asociaciones.

CLASES DE RELACIONES

Una relación será de agregación siempre y cuando pueda decirse de un objeto que "es parte de" otro. Así entre dos objetos coche y motor existirá una relación de este tipo. Las relaciones de agregación no son reversibles; es decir, el motor es una parte del coche pero no al revés. Una relación será de herencia cuando de un objeto pueda decirse que "es un especie de" otro. Al igual que sucede con las de agregación, éstas tampoco son reversibles. De este tipo de relación se han visto múltiples ejemplos a lo largo del curso. El resto de las relaciones reciben el nombre genérico de asociaciones.

Es preciso señalar que, en muchas ocasiones, es difícil poder encasillar una relación cualquiera en uno u otro tipo. El resultado final dependerá, fundamentalmente, de la experiencia y gustos de la persona que construye la aplicación.

Las distintas responsabilidades de la aplicación han de ser distribuidas, siempre, entre los diferentes objetos de la misma de tal manera que todos necesiten, por tanto, relacionarse y colaborar entre si para llevar a buen término las distintas tareas que se les encomiendan.

EL MEJOR AMIGO DEL HOMBRE

A lo largo de este curso, se han visto ejemplos de las dos primeras formas de relación, sin embargo siempre se ha quedado en el tintero hacer que dos objetos distintos, no siendo uno parte del otro, se relacionen de alguna manera. Hay muchas formas de conseguir esto; sin embargo, en este artículo se va a tratar con la más simple de todas: el trabajo con punteros. Hacer que esos punteros siempre tengan un valor válido es lo que se llama integridad relacional.

Imagine que, para implementar la relación que existe entre un perro y una persona se colocan dentro de ambos objetos sendos punteros, uno al otro. Cuando el puntero que va de la persona al perro tenga valor NULL querrá indicarse que el primero no tiene amigo, mientras que si es al revés será el perro el que no tenga dueño.

Suponga que desde otra parte de la aplicación alguien mata al perro mediante un delete (un camión, por ejemplo). Si el animal, antes de morir no informa de alguna manera a su amo, si es que lo tiene, de que se muere, el puntero que dentro de la persona señala al perro se quedará apuntando a posiciones de la memoria que han dejado de existir. No hace falta explicar los problemas que pueden derivarse de ello.

UNA GRAN AMISTAD

Siempre que se crea un objeto cualquiera, se ejecuta si es que está definido, el constructor de la clase a la cual pertenece. No cabe duda de que este es un buen momento para hacer que las relaciones entre dos objetos comiencen a tener vigencia.

Por ejemplo, uno de los parámetros pasados al constructor de la clase persona, podría ser la dirección del perro que va a ser su compañero y amigo (NULL en el caso de que no se considerara). El puntero atributo interno, que señala a un objeto perro será igualado a este valor parámetro. Este mismo valor, si no es NULL, puede aprovecharse para informar al perro, de que ya tiene un nuevo dueño. Algo completamente análogo podría decirse del constructor de la clase Perro:

```
class Persona {
 public :
  ...
  Perro *mi_perro;
  Persona (const char *n,Perro *p) {
    ...
    mi_perro = p;
    if (mi_perro != NULL)
      mi_perro -> mi_amo = this;
    ...
    }
  ...
};

class Perro {
 public :
  ...
  Persona *mi_amo;
  Perro (const char *n,Persona *p) {
    ...
    mi_amo = p;
    if (mi_amo != NULL)
      mi_amo -> mi_perro = this;
    ...}
  ...
};
```

CUANDO UN AMIGO SE VA

Cuando un objeto deje de existir se ejecuta, si es que está definido, el destructor de la clase a la que pertenece. Esta circunstancia, puede ser aprovechada para garantizar la integridad relacional entre todos los objetos de una aplicación.

Así, siguiendo con el ejemplo, podríamos usar las primeras líneas de código del destructor de la clase perro para igualar a NULL el puntero que dentro del dueño señala hacia él; claro está, si es que existe. Algo análogo podría decirse del destructor de la clase Persona:

```
class Persona {
 public :
  Perro *mi_perro;
  ...
  ~Persona () {
  if (mi_perro != NULL)
  mi_perro -> mi_amo = NULL;
  ...
  }
 ...
};

class Perro {
 public :
  Persona *mi_amo;
  ...
  ~Perro () {
  if (mi_amo != NULL)

      mi_amo -> mi_perro = NULL;
      ...
    }
 ...
};
```

CAMBIANDO DE AMIGO

Si la relación de amistad existente entre personas y perros está representada por un puntero dentro de los objetos de la primera clase a los de la segunda, bastaría con cambiar el valor de éste para reflejar que una persona ha cambiado de perro.

Si hacemos esto, sin más, tanto el perro nuevo como el antiguo estarán señalando a objetos de tipo persona que no se corresponden con la nueva situación. La coherencia de las relaciones se deshace.

Para que una persona cambie de perro no basta, por tanto, con cambiar el valor del puntero "el_perro", que hay dentro de los objetos persona sino que es necesario, además, modificar el de los puntero "mi_amo", que hay dentro tanto del antiguo como del nuevo perro.

Para hacer esto podríamos construir un método dentro de las clases relacionadas. Así, por ejemplo:

```
class Persona {
  public :
  Perro *el_perro;
  inline void cambiar (Perro *p) {
  if (el_perro)
    el_perro -> mi_amo = NULL;
  el_perro = p;
  if (el_perro)
    el_perro -> mi_amo = this;
  }
  ...
};

class Perro {
  public :
  Persona *mi_amo;
  inline void cambiar (Persona *p) {
   if (mi_amo)
     mi_amo -> el_perro = NULL;
   mi_amo = p;
   if (mi_amo)
     mi_amo -> el_perro = this;
  }
  ...
};
```

¡CUIDADO!

Es evidente que, dada la forma en la que, hasta ahora, se ha tratado el asunto, los punteros que sostienen la relación han de ser públicos. De esta manera es fácil acceder, a uno u otro desde cualquier parte de las clases relacionadas.

Sin embargo, también puede ser accedido desde el exterior de ambas y, por tanto, sus valores pueden ser arbitrariamente modificados, echando por tierra todas sus buenas intenciones sobre integridad relacional.

No parece apropiado, por otro lado, aunque no hay nada escrito sobre gustos, que se pongan métodos para poder conocer y modificar algo tan natural como un canal de relación del objeto.

Se va a tratar de construir una forma de mantener la integridad relacional que sean en todo análogos a los punteros contemplados hasta ahora y que, por otro lado, su modificación asegure que la integridad relacional sigue manteniéndose.

Hay que precisar que esto es sólo un ejemplo sobre como manejar la integridad relacional y que otras personas pueden tener otras ideas al respecto. Eso es, precisamente, lo que nos hace grandes.

RELACIONES MÚLTIPLES

El código expuesto tanto para el constructor como para el destructor como para el método cambiar puede llegar a complicarse mucho, si se llega a considerar relaciones distintas a la simple uno contra uno. Imagine, por un momento, que la persona no tiene un sólo perro sino muchos y que por tanto, en vez de tener un puntero a un objeto de tipo perro tiene un array de ellos. El destructor de la clase perro, ahora, tiene que buscar su dirección entre los punteros referenciados por el amo, eliminarse y reconstruir, después, el array. Si, además, consideramos que un perro puede tener varios dueños esta misma complejidad la encontrará en el destructor de la clase persona.

El hablar de array es simplemente indicativo, ya que las relaciones entre muchos objetos, al igual que las de uno contra uno, pueden ser implementadas de muy diversas maneras no necesariamente mediante arrays.

AUTOMATIZANDO LA TAREA

Piense, ahora, en otros dos objetos de otras dos clases cualesquiera entre los que también pueda establecerse una relación de uno a uno. Las tareas que se realizan tanto en constructor como en destructor son, al menos en lo que respecta al mantenimiento de la relación, completamente análogas. Lo ideal sería, sin duda, llegar a automatizar los procesos de construcción y destrucción de esas relaciones y hacerlos completamente trasparentes para el

Figura 1

```
class R_persona_a_perro {
 protected :
        Persona *CONTINENTE;
public :
        Perro *__ptr;
            ...
        inline void owner (Persona *p) { CONTINENTE = p; }
        ~R_persona_a_perro () {
           if (__ptr != NULL)
              ptr -> mi_amo.__ptr = NULL;
        }
        inline Perro *operator = (Perro *p) {
              ptr = p;
              if (__ptr != NULL)
                 __ptr -> mi_amo.__ptr = CONTINENTE;
        }
        inline Perro *operator -> (void) {
              return (__ptr);
        }
        ...
};
class R_perro_a_persona {
protected :
        Perro *CONTINENTE;
public :
        Persona *__ptr;
        ...
        inline void owner (Perro *p) { CONTINENTE = p; }
        ~R_perro_a_persona () {
              if (__ptr != NULL)
                 ptr -> el_perro __ptr = NULL;
         }
        inline Persona *operator (Persona *p) {
             ptr = p;
              if (__ptr != NULL)
              __ptr -> el_perro.__ptr = CONTINENTE;
        }
        inline Persona *operator -> (void) {
             return (__ptr);
        }
        ...
};
class Persona {
public :
        R_persona_a_perro el_perro;
        Persona (Perro *p = NULL) {
              ...
             el_perro.owner (this);
             el_perro = p;
             ...
        }
        ~Persona () { /* Vacio */ }
        ...
};
class Perro {
public :
        R_perro_a_persona mi_amo;
        Perro (Persona *p = NULL) {
              ...
             mi_amo.owner (this);
             mi_amo = p;
             ...
        }
        ~Perro () { /* Vacio */ }
...
};
```

resto del código de esos métodos.

Para automatizar el proceso de construcción y destrucción de relaciones se va a hacer uso de ciertas propiedades del constructor y del destructor que, hasta el momento, se han preferido obviar.

LOS ATRIBUTOS OBJETO

Si dentro de una clase se definen como atributos de la misma objetos de otras (no punteros a ellos), dentro del constructor del primero se ejecutan los de los demás. El constructor ejecutado será, o bien uno por defecto, o bien aquel que usted especifique en la lista de inicialización, pero siempre se ejecutan. El orden en el que se construyen esos objetos atributo es el mismo, que el que tienen dentro del fichero cabecera de definición de la clase que los contiene.

Lo mismo puede decirse del destructor. Siempre que se ejecuta el destructor de una clase se ejecutan, también, los destructores (si es que están definidos) de todos los objetos que haya definidos dentro del primero. El orden en el que se destruyen esos atributos objeto es el inverso al que tienen dentro del fichero cabecera de definición de la clase, es decir, el orden inverso al que se empleo durante la construcción.

UNA RELACIÓN AUTOMÁTICA

Se va a usar, por tanto, un objeto atributo para mantener la integridad relacional. Según lo visto anteriormente, cuando se ejecute el destructor de la clase que lo contiene se ejecutará, también, el de ésta, momento que tendrá que ser aprovechado para garantizar coherencia entre ambos lados de la relación.

El objeto atributo, que llevará la relación tendrá que pertenecer a una clase, en la que uno de sus atributos internos sea el puntero que hasta ahora hemos manejado, y que ahora camuflamos.

Por otro lado es objetivo nuestro, que el comportamiento de ese objeto sea, en todo, homólogo al del puntero que camufla; es decir, que pueda ser igualado, comparado y tratado como tal. Cosas, todas ellas, que pueden ser fácilmente conseguibles a través de la sobrecarga de operadores.

Mención especial merece la sobrecarga del operador "=". Este operador es el que nos permitirá cambiar con facilidad el valor del puntero camuflado, pero consiguiendo como efecto colateral que el puntero del otro lado de la relación también se entere del cambio.

Para conseguir un comportamiento completo como puntero habría que sobrecargar, además, el operador "==", el operador "!=", el operador "->" y otros cuantos más que, en principio, no se considerarán (figura 1).

DUPLICIDAD EN LOS NOMBRES DE RELACIÓN

Piense, una vez vista la figura 2, en cómo habrían de ser las clases que sostuvieran la relación entre otros dos objetos, de otras dos clases cualesquiera.

Todo sería exactamente igual que hasta ahora, a excepción de los nombres de las clases, que representan la relación (R_persona_a_perro, R_perro_a_persona), y los de los objetos de esas clases, que la llevan a cabo (el_perro y mi_amo).

Imagine que, por otro lado, fuera necesario establecer dos relaciones distintas entre esas mismas clases. Los nombres que hasta ahora se han elegido para las clases relación basadas en los de las clases que relacionan no servirían ya que, en este caso, se tendría una duplicidad en los mismos, a pesar de contener diferente tipo de información.

Para nombrar, por tanto, a una clase relación, es necesario tener en cuenta los nombres de los objetos que la van a sostener. Es decir nuestras relaciones deberían haberse llamado, por ejemplo, R_persona_elperro_a_perro_miamo o R_perro_miamo_a_persona_el_perro.

Incluir el nombre del objeto relación en el de la clase relación, asegura que no habrá duplicidad en los segundos, ya que dentro de una misma clase no puede haber dos objetos con la misma denominación.

TIPOS ANIDADOS

Dentro de C++ se pueden definir tipos de datos dentro de otros. Así, por ejemplo, es posible declarar una clase dentro de otra o definir un enumerado dentro de una clase. Los tipos así declarados reciben el nombre de tipos anidados.

```
class CLASEA {
public :
  class CLASEB {...};
  ...
};

void main (void)
{
  CLASEA objeto1;
  CLASEA::CLASEB objeto2;
  ...
}
```

El nombre de un tipo anidado visto desde el exterior de la clase donde se definió es el mismo, pero precedido por el de esa clase más dos puntos:

Ya que los nombres de las clases que definen la relación entre otras dos están tan relacionadas no sólo con los nombres de las clases relacionadas (Persona y Perro), sino también con la de los objetos que la sustentan (el_perro, mi_amo), podrían definirse como tipos anidados.

EL LENGUAJE DE LAS MACROS

Las clases que sustentan una relación uno a uno tienen un código completamente análogo, variando únicamente los nombres de las variables implicadas. Por tanto, no sería mala idea implementar la relación mediante una macro parametrizable, y dejar que fuera el compilador quien escribiera el código por usted.

Ésta es la técnica que se ha empleado en el ejemplo de este mes. El artículo se acompaña con una macro, que permite la construcción automática de relaciones. Los parámetros de la misma son, por orden: Nombre de la clase propietaria de la relación, nombre del objeto que dentro de ésta va a sustentar la relación, nombre de la clase referenciada y, nombre del objeto dentro de aquella que sustentará la relación inversa.

La macro ha sido dividida en tres. La primera habrá de incluirse dentro del fichero cabecera de la clase origen de la relación, mientras que la otra tendrá que incluirla en las primeras líneas del código .cpp donde describe el cuerpo de aquella (figura 2).

Figura 2

```
class Persona {
public :

  RELACION1_1(Persona,el_perro,Perro,mi_amo)
   el_perro;
   Persona (Perro *p = NULL) {
   RELACION_INICIAR (this); el_perro = p;
   }
   ~Persona () { /* Vacio */ }
   ...
};

class Perro {
          public :

RELACION1_1(Perro,mi_amo,Persona,el_perro)
mi_amo;
  Perro (Persona *p = NULL) {
  RELACION_INICIAR (this); mi_amo = p;
  }
  ~Perro () { /* Vacio */ }
  ...
};
```

Por último es necesario incluir una tercera macro en las primeras líneas de los constructores, que se encargue de informar a objetos relación, sobre quien en su objeto propietario para que ellos puedan emplear esa información, en el manejo de la integridad relacional.

Las relacciones entre los objetos son de agregación, de herencia y de asociación

OTRAS RELACIONES

Las asociaciones entre objetos se dividen, básicamente, en cuatro grandes grupos: uno a uno (una persona y un perro), uno a muchos (una persona y sus dedos), muchos a uno (los empleados de una empresa y ésta) y, por último, muchos a muchos (un grupo de amigos).

En las primeras tanto un lado como otro de la relación están representados por punteros. En las segundas el lado origen es una lista de punteros a objetos del tipo destino, mientras que el lado destino es un puntero al lado origen. En las terceras es al revés que las anteriores; mientras que en las cuartas tanto en un lado como en el otro son listas de punteros a objetos.

Implementar todas estas relaciones puede ser complicado. Vaya pensando como hacerlo.

NOTA FINAL

Existen otras muchas formas más complicadas de llegar a implementar las relaciones 1 a 1 de objetos. Sin embargo, se ha estimado que ésta es una forma bastante sencilla y fácil de seguir, y que puede servir como guía para investigar otras. Es difícil que todas las personas que lean este artículo coincidan con sus postulados. Recuerde que no es algo establecido por ANSII, sino sólo una forma de solucionar algo que el C++ no tiene contemplado, al menos de momento.

Es necesario, además, que no siempre que dos objetos estén relacionados entre si, sea imprescindible crear toda esta estructura de relaciones. Usted será el que juzgue, si es o no oportuno.

COMO COMPILAR EL EJEMPLO

El artículo sólo viene con un ejemplo. Para compilarlo basta cargar el proyecto asociado y seleccionar Run. El ejemplo emplea un lenguaje de macros estándar ANSI, por lo que no debe haber problemas, para su compilación en cualquier plataforma.

Investigue, cree sus propias clases, y relaciónelas, no hay mejor forma de probar las posibilidades de esta forma de trabajo.

LA SIGUIENTE ENTREGA

En la siguiente entrega, se dará por finalizado el curso presentando un resumen de las cosas más importantes de todos y cada uno de los puntos tratados. ■

CORREO DEL LECTOR

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relaccionado con la programación.

P *He observado que en algunos compiladores de C o C++ (Watcom, Symantec), viene la opción para generar código específico para el Pentium, ¿cuáles son las instrucciones específicas del Pentium con relación a un 486?, ¿se nota mucho incremento en la velocidad del código? Gracias.*

Miguel Ángel Parra Martín
(Málaga)

R Los compiladores no se refieren a que si utilizas esa opción se genere código exclusivo para el Pentium, lo que hacen es "recolocar" y ordenar las instrucciones para poder aprovechar mejor la potencia del Pentium de procesar simultáneamente varias instrucciones, es más, suelen recomendar que aunque el programa vaya a ejecutarse sobre un 386 ó 486, se utilice esta opción, pues a estos procesadores les da igual como vengan las instrucciones y si se ejecuta sobre un Pentium, pues algo se ha ganado (hablan de un diez por ciento de mejora en la velocidad).

P *Estoy intentando realizar transferencias DMA con el compilador Symantec C++ que regalaron, utilizando el modo protegido. Como la dirección de los datos DMA debe encontrarse en el primer mega de memoria (20 bits), tengo que reservar memoria en esa zona. ¿Cómo lo realizo? Gracias por todo.*

Gonzalo Aguilera
(Ávila)

R Las transferencias DMA se pueden realizar en cualquier dirección de los 16 Mb inferiores, ya que su dirección es de 24 bits (16 bits en el registro de dirección y 8 en el de página), no de 20 bits como comentas. Pero para pedir memoria "baja" en modo protegido existe la función DPMI 100h: Allocate Dos Memory (Int 31h 0100h).

P *Mi problema es el siguiente: compré el número 4 en el que regalaban el compilador Symantec C++ incluido en un CD-ROM, yo no dispongo de un lector de CD, pero puedo acceder a uno de un amigo. ¿Cómo intercambio el compilador desde CD a disquetes?, creo que originalmente venía en ellos, por lo que debe ser factible el proceso inverso. De esta manera podré instalarlo en mi ordenador y disponer de una copia personal. ¿Instalaciones parciales? ¿Información adicional sobre las librerías del compilador? Un saludo.*

Fernando Alías Lacámara
(Zaragoza)

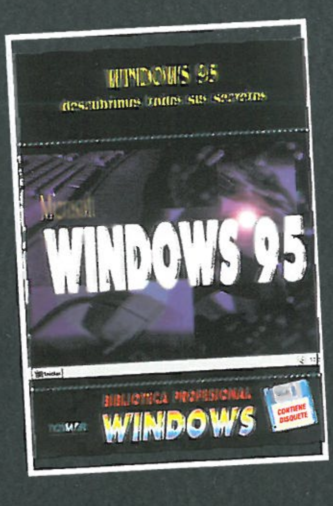
R Tiene razón, es posible volverlo al soporte en discos. Lea o imprima el fichero INSTALL.INF que se encuentra en el directorio \SYMANTEC\SETUP, en él aparecen los ficheros contenidos en cada uno de los 17 discos originales. Sólo tendrá que copiarlos para crearse otra vez la copia en disquetes, teniendo en cuenta que en el disco 1 también se incluyen los ficheros auxiliares para la instalación (el resto de los ficheros no listados).

Mediante el programa de instalación se puede elegir la parte a instalar, sin necesidad de introducir todo en el disco duro; por ejemplo, si sólo se va a desarrollar para MS-DOS no se tendrán que instalar los componentes de Windows. Para localizar información relacionada con las librerías del compilador, use los ficheros de ayuda tipo Windows que acompañan al compilador (Help OnLine).

P *Hola, llevo un tiempo intentando enlazar código realizado en C con código de Clipper, dispongo de Clipper 5.01 y Borland C++ 3.0, pero hasta el momento no lo he conseguido. En la poca información que acompaña a Clipper referente a este tema, siempre indica al compilador de C de Microsoft y algunos compañeros me comentan que no es posible realizarlo con Borland. ¿Que hay de cierto en todo esto? Intento seguir los pasos del artículo aparecido en el número 1 sobre este tema pero no lo consigo en mi caso. Si pueden comentarme algo, se lo agradecería sobremanera.*

María Martínez Sainz
(Burgos)

R La duda entre Microsoft y Borland es totalmente infundada, pues Borland es casi totalmente compatible con Microsoft (además de su modo propio de trabajo). Por otro lado, es completamente posible el enlace entre C (Microsoft, Borland o Symantec) y Clipper (más fácil a partir de la versión 5.0). No envía una copia de lo que intenta enlazar, por lo que no puedo saber en donde está el error. Intente empezar el enlace sin enviar parámetros y luego vaya añadiéndole complejidad. Como referencia, utilice el artículo del número 1, pues compila y funciona perfectamente sin ninguna modificación. Si continúa con los problemas, envíenos algo de su código y le contestaremos de manera privada.



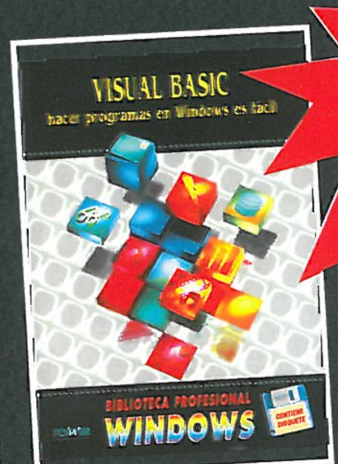
El disquete que acompaña al libro incluye utilidades para WINDOWS 95.

COREL 5
el programa de diseño que usan los artistas
BIBLIOTECA PROFESIONAL
WINDOWS
CONTIENE DISQUETE

El CD-ROM que acompaña al libro incluye una versión reducida de COREL DRAW-5.



El disquete que acompaña al libro incluye utilidades MULTIMEDIA.



El disquete que acompaña al libro incluye una demo de VISUAL BASIC

LOS LIBROS DE INFORMÁTICA MAS ACTUALES AL MEJOR PRECIO

POR SÓLO 1.695 Ptas.
con IVA incluido

TÍTULOS PUBLICADOS

1 WordPerfect 6.0
2 Comunicaciones entre ordenadores
3 MS-DOS 6.2
4 Lotus 1-2-3-4.01
5 Multimedia
6 WinWord 6.0
7 Lenguaje C/C++
8 Diccionario de términos
9 Excel 5.0
10 Seguridad informática
11 AutoCAD 12
12 Todos los secretos de Windows
13 PowerPoint 4.0
14 Visual Basic
15 QuarkXpress 3.2
16 Qué ordenador comprar
17 FoxBase Pro 2.5
18 Windows 3.11 para Trabajo en Grupo (I)
19 Windows 3.11 para Trabajo en Grupo (yII)
20 Corel 5.0
21 Paradox 4.5
22 Quattro Pro 5.0
23 Photoshop 2.5
24 Access 2.0
25 Harvard Graphics 3.0
26 dBase para Windows
27 Borland C/C++ 4.02
28 Redes informáticas
29 ToolBook 3.0
30 Windows 95

Solicite catálogo al Telf.: 91-741 26 62



COMMUNICATIONS, S.R.L